ON THE CORRECTNESS OF GOSSIP-BASED MEMBERSHIP PROTOCOLS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

André Alain Patrick Allavena

January 2006

ON THE CORRECTNESS OF GOSSIP-BASED MEMBERSHIP PROTOCOLS

André Alain Patrick Allavena, Ph.D.

Cornell University 2006

The importance of scalability and fault-tolerance in modern distributed systems has led to considerable research in multi-cast protocols using *gossip*. In a gossip protocol, each node forwards messages to a small set of "gossip partners" chosen at random from the entire group membership. By discarding the strong reliability guarantees of traditional protocols in favor of probabilistic guarantees, gossip protocols can deliver greater scalability and fault tolerance. In early gossip algorithms, partners were chosen uniformly at random from the entire membership, limiting scalability because of the resources required to store and maintain complete membership views at each node. Later protocols avoided this issue by storing much smaller random subsets of the membership at each node, and choosing gossip partners only from these *local views*. Such protocols are subtle: at least some local views must change in response to group membership changes in order to preserve connectivity and performance guarantees. While these protocols have been the subject of much simulation and analysis, formal proofs of key properties – in particular the probability of partitioning – have remained elusive.

In this thesis we give a new scalable gossip-based algorithm for local view maintenance, together with a proof that the expected time until a network partition is at least exponential in the view size and the size of the departing set. We develop probabilistic bounds on the in-degree (hence the load) of individual

nodes, and argue that protocols lacking our *reinforcement* component eventually converge to star-like networks, whose connectivity depends on a small set of overloaded nodes. We argue that the undirected connectivity graph is an expander, for which application-level gossip multi-cast protocols will converge rapidly. An analysis of the membership system under heavy churn yields a lower bound on the amount of communications required per round. Finally, we offer some arguments supporting the experimental fact that the elements of the local views – although not a uniformly random sampling of the set of nodes in the system – have a high degree of randomness and suggesting that the state of the system after $\mathcal{O}(\ln n)$ iterations is independent of the initial state.

BIOGRAPHICAL SKETCH

André Allavena attended school in the Principality of Monaco. After completing secondary school he spent one year of intensive study in classical violin at the Académie de Musique Prince Rainier III of Monaco whilst attending the Science University of Nice, France, part-time. Following this he spent two years in "classes préparatoires aux Grandes Écoles d'ingénieurs" at Lycée Masséna in Nice, before receiving his Diplôme d'Ingénieur des Arts et Manufactures from École Centrale Paris in 2001. During the final year of his curriculum at École Centrale, André pursued external studies at Cornell University, completing a Master of Engineering degree in Computer Science. After completing his Ph.D. in Computer Science, André will pursue post-doctoral research at the University of Waterloo, Ontario, Canada.

À mes parents Béatrice et Michel Allavena, à mes sœurs Sylvie et Nathalie,

Et à ma Rachel chérie.

ACKNOWLEDGEMENTS

I would like to thank my advisers, Dr. John E. Hopcroft and Dr. Alan Demers, for their advice and guidance, always being available to answer my questions, and for pointing out the obvious and not so obvious. Likewise, I would like to thank Dr. Éva Tardos and Dr. Ken Birman for their guidance during the course of my Ph.D., as well as Dr. Robbert van Renesse and Dr. Gün Sirer for the intellectual guidance and research collaborations. Dr. Graeme Bailey similarly should be thanked for the sage advice and help over the course of my degree, as well as the fantastic musical accompaniment. Acknowledgment must also go to the administrators and technical support personnel of the Department of Computer Science, particularly Stephanie Meik, Becky Stewart and Kathy Carpenter. Thanks also to my special committee members in the Department of Music, Dr Kia-Hui Tan and Dr. Steven Stucky. Last but not least, I would like to thank my fiancée Rachel for her love and care.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Gossip Algorithms

1.1.1 An Illustrative Example

The caterers for the Christmas party of Cornell's Computer Science Department are caught in a snow storm at the outskirts of Ithaca and will not be able to reach the party site. A food-less holiday party can be avoided by asking resourceful students, staff and faculty to improvise a pot-luck dinner. The organizers would like to contact everyone quickly so as to give them enough time to cook. How can everybody be contacted quickly enough?

Organizers could easily contact everyone using a list of phone numbers, phoning the first person on the list, then the second person, and so on. However, since the department is quite large, this process will take a long time. The people at the end of the list will receive the call too late, so we will miss out on Vicky Weissman's chocolate mousse. The time required to reach everybody is linear in the number of people who must be reached.

This time can be shortened by forming a "calling tree," as represented in Figure 1.1. However, such a tree would need to be built in advance. Furthermore, the tree would need to be rebuilt each time a member was unavailable, whether long term such as a student graduating, or short term such as a visit to the shopping mall. In Figure 1.1, if node B is unreachable, none of the nodes below it will receive the message originated at node A. The unavailability of a single member may prevent a significant number of other members from receiving the message.

This is a serious reliability issue.

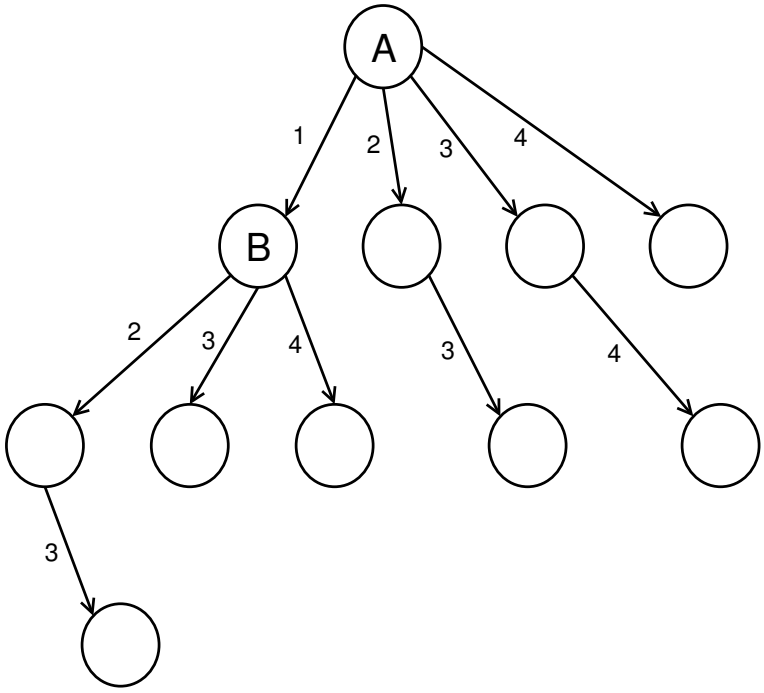


Figure 1.1: Calling Tree

The numbers represent the time at which the edge will be used for a call.

If node B dies, none of the four nodes below it will receive the message.

In this introduction we will first present gossip algorithms, a class of algorithms that solve the problem described above. In large computer networks, group membership maintenance can be an issue. Some applications designed for very large networks like peer-to-peer systems have a weak notion of membership. Others, like gossip-based systems, require a membership management protocol. We introduce the idea of using a gossip algorithm to solve the membership maintenance problem for gossip-based systems.

1.1.2 Gossip Algorithms

Reliability is linked to scalability: the more nodes, the more failures. Thus a system will have poor scalability if it incurs a failure recovery cost proportional to the number of failures. Conventional systems that provide strong reliability guarantees do not scale well because of this problem. To avoid these difficulties, other approaches, known as gossip (or epidemic) algorithms have been introduced [1, 5, 18, 35, 36, 53, 78, 76, 77, 39]. Strong reliability properties are traded for probabilistic ones. This is required since for most distributed systems, both the frequency of infrequent events (failures) and their recovery time grows at least linearly in the size of the system, triggering a costly $\Omega(n^2)$ phenomenon.

In the Christmas party example, the gossip algorithm would work as follows. In each round, every person who already knows the message would choose a person at random and phone them the message. Initially, only the organizers and the first person they called know of the message. They each call somebody during the second round. Chances are that they will call two new, different persons. During the third round, four people will be making phone calls, reaching presumably four new people. Eight persons now know the message. It is not difficult to see that with high probability, after $\mathcal{O}(\ln n)$ rounds, every person has received the information, assuming the choice of the person to call is uniformly random [7, 65]. This will be the case even if some of the phone calls are unsuccessful or if some people refuse to forward the message as they are not coming the to Christmas party. Frieze and Grimmett, and Pittel [26, 68] showed that the number of rounds is $\log_2 n + \ln n + o(\ln n)$ with high probability. In contrast, the number of rounds in our example was n when the organizers called everybody in turn and is on the order of $\mathcal{O}(\ln n)$ when the calling tree is balanced. It cannot be done faster than

$\Theta(\ln n)$.

Gossip algorithms achieve reliability and speed of delivery at the price of extra messages. While in the case of the tree, there were $n-1$ messages sent, the gossip algorithm sends $\mathcal{O}(n \ln n)$ messages. When the number of rounds before which everybody should have received the messages is $\mathcal{O}(\ln n)$, the minimum number of messages to be sent is $\Theta(n \ln \ln n)$ and can be achieved by some strategies deciding when nodes stop forwarding messages [45]. Reference [18] also achieved that minimum. More information can be found in the following old surveys: [38, 23].

1.1.3 Types of Gossip Algorithms

The algorithms described above are uniform gossip algorithms [65, 7]. Every node chooses communication partners by selecting uniformly at random among the set of nodes in the system. However, other distributions are possible. One of the drawbacks of uniform gossip is that network topology is not taken into account: since the message destinations are chosen uniformly at random, most of the messages are sent to distant nodes. This has both theoretical and practical implications.

Having the probability distribution of communication between nodes be inverse polynomial in their distance instead of uniform ensures that most messages are sent to nearby nodes, thus significantly reducing network load [49]. A consequence of this distribution is that information travels a distance d in a time polylogarithmic in d. In other words, the message spreads from the source outwards, reaching the closest neighbors first instead of reaching the nodes in an completely disordered way. This last property has given the name of "Spatial Gossip" to this gossip variant where the communication probability distribution is inverse polynomial in the distance. Later work [48] shows that uniform gossip is incapable of finding approx-

imate solutions to the minimum spanning tree or the resource location problem with short messages, whereas spatial gossip can.

A practical approach to avoid sending large numbers of superfluous messages far away is to build a locality-based hierarchy [35, 77, 54, 27, 30]. The membership of a given cluster is fully known to the members of that cluster but not necessarily to the members outside of that cluster.

1.2 Large Networks

Membership Management is a complex issue for large systems and has received a considerable amount of work. Some collaborative systems like Gnutella [31] or Freenet [25, 15] have several million members. Most of these collaborative systems fall into the category of peer-to-peer systems. The architecture is not a traditional server - client architecture. Instead all nodes are equal, acting as both client and server, and all tasks are distributed.

1.2.1 Peer-to-Peer Systems: Object Sharing

A lot of work has gone into peer-to-peer systems providing file sharing capabilities. There the main issue is how to locate existing objects, that is, how to route requests efficiently. Some early systems, like Gnutella and Freenet, are completely unstructured. They maintain several copies of each object. A copy of a desired object is located by flooding the network with a request for the object. An alternative to flooding is to forward requests randomly, in essence making each request do a random walk in the network until it reaches a node with a copy of the desired object. Some other peer-to-peer systems are structured in the sense that some central server(s) help(s) localize the requested objects [20, 6]. Many others

have structured address spaces (typically a distributed hash table) that enables peers to forward requests appropriately and to efficiently locate objects in the absence of any centralized service: Chord [75], Pastry [72], Tapestry [82], Can [69], Viceroy [61], Kademlia [63] and others [73]. See [55] for a survey of various peer-to-peer schemes. These systems have very little state stored at every node, typically $\mathcal{O}(\ln n)$, and have provably logarithmic routing time in the absence of failures. Reference [2] provides bounds on the routing time in the case of failures.

Kelips [34] revisited the trade-off between routing time and memory consumption, arguing that even a memory footprint of $\mathcal{O}(\sqrt{n})$ was manageable. With a $\Theta(\sqrt{n})$ memory footprint, the routing time is constant (when the routing tables are reasonably accurate). Other work on one-hop distributed hash tables includes [33, 32, 71]. See [81] for an analysis of the trade-offs between memory footprint and network diameter.

1.2.2 Communication

The "raison d'être" of networks is communication. Networks enable people and computers to communicate, cooperate or interact. Two of the first and simplest communication applications were broadcast and multicast: sending a message to all the nodes in the network (broadcast) or to many nodes in the network (multicast). The Christmas party pot-luck call, mentioned at the beginning of this introduction, is a broadcast. Multicast and broadcast have been extensively studied: see [23, 38, 70, 3, 5, 24, 35, 22] and [17] for a survey.

Multicast capabilities can also be added to other existing systems. For example, the peer-to-peer systems we mentioned above are designed with object sharing in mind but can easily be used to provide multicast capabilities [9, 11]. The structure

permitting routing in the peer-to-peer system is used for the multicast. This structure can be made locality aware, improving the performances of the localization /object routing scheme by preferentially routing to nearby nodes instead of distant ones [8]. Making the structure locality aware also improves the performances of the multicast operation [10] using it.

Advances in technology have made possible new kinds of networks. Nodes can be mobile, changing location several times a day. Such networks are known under the name of mobile ad-hoc networks (manet). Or nodes can be very small, immobile and limited in power, like monitoring sensors. These constraints can lead to high sensitivity to load and congestion. Also, sending long distance messages is more problematic than in wired networks, since wireless range is limited and nodes are able to communicate directly only with nearby nodes. See [79] for a survey of reliable broadcast protocols specifically designed for mobile ad-hoc networks. Node mobility means that routes often change in these networks. Broadcast is often used as a building block for the routing algorithm. Of interest among others are [56] and [37], which use a gossip-based broadcast algorithm to find routes.

Some applications are intertwined with the communication layer, aggregating messages to compute an average or a summary before forwarding the relevant piece of information. Astrolabe [77] is a large system with an administrator-built hierarchy designed to collect large-scale system state and aggregate predefined attributes. The work of Kempe et al. [47] shows how to use gossip to compute simple aggregates like the average of the value held by the members of a system using a gossip algorithm. But the largest amount of work has dealt with networks of small sensor nodes typically used for monitoring purposes [12, 21, 40, 58, 57, 59, 44].

1.3 Membership Management for Gossip Algorithms

1.3.1 Motivation

Most peer-to-peer systems store some membership information, if only because that membership information is necessary in order to provide other functionality liking routing. Membership maintenance is not a primary goal, rather a byproduct. However some applications might want a membership primitive. Gossip algorithms, in particular, require knowledge of the membership in order to select random peers to communicate with. Other applications may also require the ability to select a peer at random. Keeping the whole membership at every node might not be feasible for memory consumption reasons. Even if it were, keeping it accurate is challenging, especially when the number of nodes leaving and joining is very high. Furthermore, a small, partial but correct list of the membership (that is, a subset of the members currently in the system) is often more useful than a complete but inaccurate list. For a node needing to communicate with few members of the system, a list containing 20 members out of which 18 are currently present in the system is certainly more useful than a list containing 3000 names with only 60 correct members. There is a 90 % chance of finding a good member in the former case but only a 1 % chance in the latter.

In some cases, however, it could be that the more nodes the better, even if the fraction of live nodes decreases. For example, consider an application where a node needs to directly reach the largest possible number of neighbors, and the cost of trying to communicate with an unreachable node is low. In that situation, it makes sense for nodes to keep a local partial membership list as large as possible, even if the list is mostly incorrect. This kind of situation is fairly atypical. Most

systems benefit from members having a smaller but more correct membership list.

1.3.2 Related Work

The idea of providing a scalable membership management for gossip-based algorithms originated in [22]. Nodes keep a view (a partial list of members currently in the system). The membership management system requires an explicit join mechanism, as well as an explicit leave mechanism. A gossip algorithm is used to exchange information about the membership changes in the system. The messages exchanged contain a list of joining members and a list of departing members that the nodes use to update their views. Periodically live nodes rejoin without signing off. When either one of the lists is longer than a predetermined size, the content of the list is randomly ordered and truncated to the maximum size.

The membership management benefits from the gossip mechanism in two ways:

1. reliability in the face of communication failures. Updates will be propagated even if some messages are lost. This includes the information lost when messages are too long and need to be truncated.

2. randomness of the membership dissemination. Information about new members is sent to random nodes, different for each new member, so the new nodes are included in the views of randomly chosen nodes.

The authors felt the membership was not exactly scalable since the view size had to be set in advance. Further work [29] yielded a variation of the algorithm named Scamp in which the view size would converge to the desired size of $\ln n$ (where n is the system size). The distribution of the view sizes was not sharp; this was remedied in [28]. Proofs of correct behavior of the membership system

were presented. They however assumed the views to be uniform samples of the membership, while simulations showed they were not.

Other gossip-based membership management algorithms that are supported by simulations have been presented [74, 42, 80]. The simulations show satisfactory protocol performance: the network does not partition and the load stays balanced. These membership algorithms work in practical settings. Araneola [64] is a gossip application level broadcast that uses one of these ([29]) and performs very well. T-Man [43] is another application using a gossip based membership management. T-Man is a topology management system that builds overlay networks characterized by some user-defined topological properties. The convergence to the desired topology is fast in the cases the authors simulated. Applications needing a "random-peer" primitive can also function with any of these gossip-based membership management. Reference [41] experimentally showed that the randomness of the view elements provided by these gossip-based membership managements, though not uniformly random, was sufficient for almost any application. However, none of these gossip-based membership algorithms are supported by a theoretical analysis justifying their good properties.

Some theoretical work on membership management for peer-to-peer networks exists, though not directly related to the work presented in this thesis. A peer-to-peer overlay network with provable connectivity, constant degree and logarithmic diameter is presented in [67] where a central server helps for most changes in membership. The nodes' views of [16] are provably uniformly random. They are obtained by swapping the end of two edges selected uniformly at random. This process cannot be implemented in a distributed way; it requires a central oracle. Reference [60] is more promising, since individual nodes can initiate the update

process. However, it requires a random walk per update and the convergence may be slow, so it is probably too costly for practical purposes.

1.3.3 Outline

In Chapter 2 we explain what gossip-based membership protocols are and present the specific variants studied in depth in this work. A high-level analysis is presented in Chapter 3: some very simple equations allow us to derive the essential traits of the behavior and evolution of these protocols. Chapter 4 contains proofs that the probability of a network partition forming is exponentially small even in the face of heavy churn. In Chapter 5, we present an analysis of the in-degree of the nodes in the system. Churn and in particular the consequences for the optimal view size are studied in Chapter 6. Some arguments explaining why views are well mixed and randomized are introduced in Chapter 7. We summarize our results and conclude this thesis in Chapter 8.

Chapter 2

Protocols

Here we describe the protocols studied in this thesis. We first define the term view, necessary to understand these protocols, then present a generic gossip-based membership protocol. Our main protocol and a variant called the swap protocol are finally introduced and explained in detail.

Definition of a View

Each node keeps a partial list of the membership set. This list, called a *view*, is a subset of the nodes in the system. As such, it may contain the node itself, but no duplicates, and is different from node to node. In our protocols, all nodes have a view of the same size; and that size is typically represented by the letter k. Also, n usually denotes the number of nodes in the system and f the fanout.

2.1 Generic Protocol

Here we present, in the interest of the reader, one of the generic forms of a gossip-based membership algorithm.

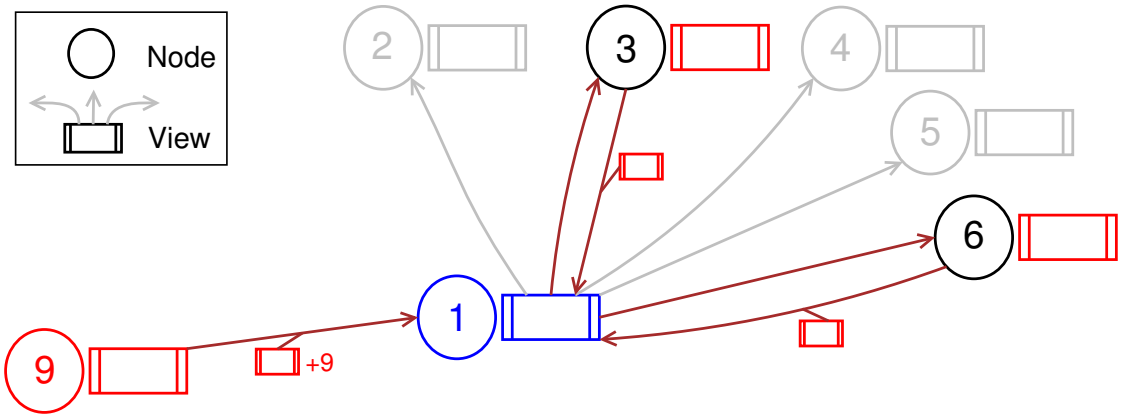
2.1.1 Protocol

At every round, each node u:

- selects at random f nodes from its current view;
- contacts these f nodes and requests their views;

- concatenates these views into a list $\mathcal{L}$;
- adds to its list $\mathcal{L}$ the nodes that request u's view during the round, if any;
- adds to its list $\mathcal{L}$ the views of the nodes that request u's view during the round, if any;
- creates its new view by selecting k elements from the list $\mathcal{L}$, not allowing duplicates.

Node u concatenates the views of the nodes with which it interacted (both the views from the nodes which u contacted and the views from the nodes that contacted u) as well as the name of the nodes that initiated a communication with u. The generic protocol is illustrated in Figure 2.1.



Node 1 has pointers to nodes 2, 3, 4, 5 and 6 in its view. It has elected to contact nodes 3 and 6. Node 9 is the only node which has chosen to contact node 1. Node 1's new view will be computed by selecting elements from the views of node 3, node 6 and node 9, as well as from the set $\{9\}$.

Figure 2.1: Graphical Representation of the Generic Protocol

The protocol can be synchronous (all nodes are updated simultaneously), loosely synchronized (nodes are updated sequentially in some random order, each node being updated exactly once per round) or asynchronous (n nodes chosen uniformly at random with replacement are sequentially updated in the round, so some nodes may be updated more than once, and others not at all). Simulations show no significant differences in behavior.

2.1.2 Potential Issues

There are three main issues any protocol faces, even in the absence of nodes joining or leaving the system:

1. The network could break into two or more disconnected components. Note that by disconnected we mean no links between components;

2. The network could become unbalanced, swamping a few nodes with a very high load instead of spreading the load more or less evenly;

3. The views could not mix well, and the system at time $t = \mathcal{O}(\ln n)$ would not be independent from the system at time $t = 0$.

Partitioning is the most severe issue the protocol faces because we cannot recover from it. We can recover from an unbalanced network; and dependency between iterations is an annoyance more than anything else, with respect to the membership management.

In later chapters we show that these conditions are unlikely to arise. Here we mention a few techniques for detecting and recovering from partitioning.

Detecting Partitioning

It is possible for the nodes of a small component to detect they are disjoint from the main component. Then, the partitioned nodes would re-join the network (the main component).

There are some techniques to compute an estimate of the size of the system. One then monitors the estimate. A sharp decrease in the estimate likely indicates the formation of a partition. The nodes experiencing the sharp decrease in size are the nodes on the small side of the partition. The estimate does not need to be very precise in order for small sets of nodes to detect they have departed from the main component. However, this technique is unlikely to detect the formation of partitions where both parts are of significant sizes.

Size Estimate

A size estimate of the network can be computed in the following ways.

1. The simplest way is to assume the existence of an application-level gossip keeping track of hop-counts. Consider the messages received by a given node. If the sources of these messages are spread across the network, most of the hop counts will be close to the diameter of the network, which should be close to the logarithm of the number of nodes in the network.

2. Another is based on the birthday paradox. Nodes look at how long it takes before seeing the same node twice. This gives an estimate of the square root of the network size. This technique is not necessarily straightforward to implement as the nodes that populates one's view are not independent uniform samples of the set of nodes.

3. Some other techniques are explored in [4].

2.1.3 Design Notes

Protocols like LwPBCast [22] or Scamp [29] require an explicit action when nodes leave the network. We believe this is unreasonable to expect, since nodes might be crashing and unable to fulfill the protocol. We decided to not explore the possibility of having nodes monitor other peers and sign them off when they are detected dead. This would be both very complex (reliably detecting failures in a distributed system is known to be difficult) and expensive with respect to communication. Instead, the protocol acts as if nodes were constantly joining, letting the nodes die out of the membership system irrespective of whether they gracefully left the network or crashed.

The view size is specified in advance; we do not set it converge to some correct size as in [29]. This is not a major issue. It is simple to make the protocol work with heterogeneous view sizes and let every node estimate the size of the network with the techniques described above.

2.2 Protocol

Here we present our chosen variant of the generic protocol and discuss the alternatives.

2.2.1 Protocol

In our variant of the generic protocol, there are two additional parameters, f, the *fanout*, and ω, the *weight of reinforcement*. Each node repeatedly updates its local

view in *rounds*; in each round, a node s will:

1. construct a list $\mathcal{L}_1$ comprising the local views of f nodes chosen at random from the local view of s,

2. construct a list $\mathcal{L}_2$ of the other nodes that requested its view during the round,

3. create a new local view by choosing k distinct elements at random from $\mathcal{L}_1$ and $\mathcal{L}_2$.[1] The reinforcement weight ω determines how much more likely nodes are to be selected from $\mathcal{L}_2$ than $\mathcal{L}_1$. If ω is 0, nodes in $\mathcal{L}_2$ are ignored; if ω is 1, we make no distinction between $\mathcal{L}_1$ and $\mathcal{L}_2$; and in the limit as ω goes to ∞, all nodes are taken from $\mathcal{L}_2$ if possible.

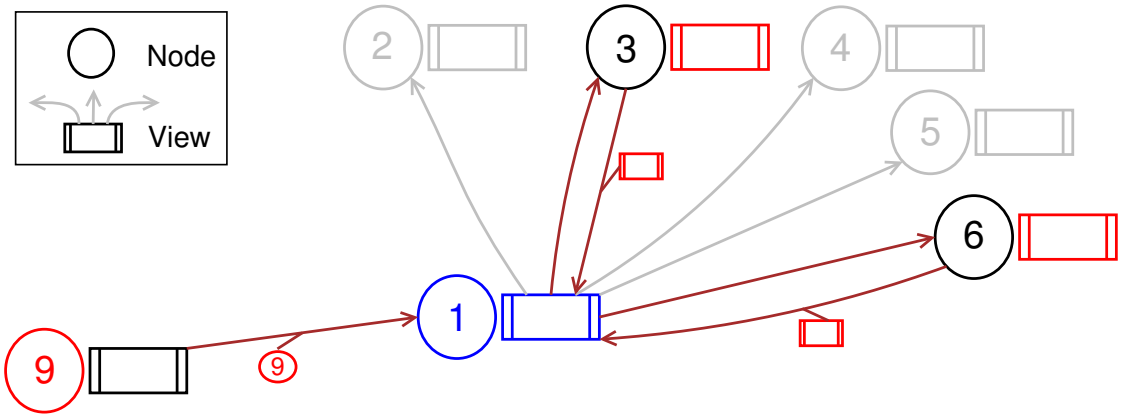
The only difference between the protocol we study and the generic algorithm is that node s does not use the views of the nodes that contacted s. See Figure 2.2 on the following page.

Joins and Leaves

A node joins the network by copying the view of some node. If the rate of nodes joining the network is small, they can all bootstrap from the same node. Since the node's view changes at each iteration, this will not lead to a major imbalance in the graph. And even if it did, the graph automatically re-balances itself as we shall see below.

Since it is unreasonable to expect nodes always to leave gracefully, the protocol has been designed not to require any termination messages from a node leaving the network, and there are no "heartbeat" or "keep-alive" messages. This solves the

[1]The exact details of duplicate removal are unimportant.



Node 1 has pointers to nodes 2, 3, 4, 5 and 6 in its view. It has elected to contact nodes 3 and 6. Node 9 is the only node which has chosen to contact node 1. Node 1's new view will be computed by selecting elements from the views of node 3, node 6 as well as from the set $\{9\}$. The view of node 9 will not be used by node 1.

Figure 2.2: Graphical Representation of Our Main Protocol

often overlooked scaling issue of the cost of nodes leaving the network. Whenever a node leaves, some dangling edges are left in the network. However, these edges gradually disappear. If the reinforcement weight is at least 1, which it should be, these edges will be purged from the network in an expected approximately k/f iterations. The presence of dangling edges in the views is not a major issue as far as connectivity is concerned. See Chapter 4 for a proof that the probability of the network partitioning into two or more disconnected components is exponentially small even in the presence of churn.

Uniformity of the Randomness in the View Selection

In steps 1 and 3 of the protocol, elements are selected at random. While we assume these choices to be uniformly random, other alternatives are worth considering. Some empirical results can be found in [41].

Removal of Duplicates

A seemingly important implementation decision is when to delete duplicates. There can be an overlap between the nodes in $\mathcal{L}$ and those in $\mathcal{L}2$. We can either remove duplicates from the list to select from, or wait until we actually have selected the same node twice, discard it, and redraw. In practice, it turns out that the probability of having duplicates is very small, and this is inconsequential; unless the number of nodes in the system is extremely small (say 16), in which cases simulations show that the latter is slightly better. More details in Appendix A on page 124.

2.2.2 Explanation of the Functioning of the Protocol

An easy way to summaries the protocol is:

$$\text{New View(u)} \;=\; \begin{array}{c}\text{SELECT}\\ \text{k nodes from}\end{array} \left(\begin{array}{c}\text{Views}\\ \underbrace{\text{requested by } u}_{\text{Mixing}}\end{array} \bigcup \begin{array}{c}\text{Nodes that}\\ \underbrace{\text{requested } u\text{'s view}}_{\text{Reinforcement}}\end{array} \right)$$

Due to its dynamic nature, the protocol automatically adapts and re-equilibrates the network (the connectivity graph) each time the connectivity graph (the directed graph built from the views) does not look like a uniform random graph with constant out degree, regardless of what caused that imbalance. There are two forces responsible for this, corresponding to the two different parts of the protocol:

The views requested by *u*. We call this *mixing*. Node u pulls views from several nodes, and almost all the nodes of the new view come from the pulled views. This process by itself, as we shall see later, ensures that the graph doesn't partition. This mixing is "pull" only. The push and push-pull alternatives are discussed in Section 2.4 on page 24.

The nodes that requested *u*'s view. We call this *reinforcement*. The idea behind reinforcement is simple: by pulling, node u learns second-hand of the existence of some nodes in the system. But u learns first-hand that the nodes that pulled it exist. Then node u positively reinforces the nodes that pulled it by adding them to the list out of which it creates its new view. Under some conditions (that the mixing part provides) this process ensures that the network stays relatively balanced. Also, the process removes older edges, thus ultimately eliminating edges pointing to dead nodes, and adds fresh edges, including some to newly joined members.

Without reinforcement the network would collapse into a star-like structure. This is why the "Weight of Reinforcement" parameter ω should be set to at least 1. Larger is better and will be either 1 or ∞ on a typical implementation.

Labeling this process *pushing* to oppose the pulling process described above would be quite misleading: it is correct that nodes are pushing some information. However, the nodes are pushing *their own names*, not the information in their views as done in the mixing (pulling) part. The crucial part is that nodes are adding their names to the pool of names; it is secondary that this is done by pushing.

These two processes will be analyzed in detail in later chapters. After connec-

tivity and load balancing, the third desirable property of the protocol is to have views which are uniform samples of the membership set and changing over time so as to emulate each node having the complete membership set, choosing different gossip targets at each iteration. Uniformity of randomness seems impossible to achieve. The views are *not* uniform samples of the membership. This will be explained in Chapter 7.

2.2.3 Graph Notation

Consider the network as a directed graph. Executing the protocol changes the graph. We are interested in studying the evolution of the graph as a consequence of running the protocol. Each node has out-degree k and variable in-degree. The edges change as follows:

Mixing The tail of edges move one node backward, possibly being duplicated or dropped: if u had edges to a, b, c and d and selects a and b to pull from, where a has edges to $a_1,\ a_2,\ a_3,\ a_4$ and b has edges to $b_1,\ b_2,\ b_3,\ b_4$, then u ends up having edges to four nodes from $a_1,\ a_2,\ a_3,\ a_4,\ b_1,\ b_2,\ b_3,\ b_4$.

Reinforcement The orientation of an edge is flipped. If v pulls from u, (meaning that there is an edge from v to u) and if u selects to reinforce v, then an edge from u to v is created.

When we consider the issue of partitioning, we always assume the graph to be *un*directed, that is, edges do not have an orientation anymore.

Edges Moving The words "moving an edge" refer to the act of changing the tail of the edge. The head does not change. For example, if node A has an edge e

pointing to node B and the edge e "moves" to node C, then C has an edge pointing to node B.

With respect to the lifespan of an edge, the fact of being "moved" has no effect. An edge does not die when being "moved" and a new edge is not created by a "move" operation. Informally, the edge e that was moved from node A to node C in the previous example is still the same edge, held by a different node.

2.3 Swap Protocol

We shall refer to the previously defined protocol as the protocol, and the one we define here as the swap protocol. The main difference between the two protocols is in the amount of mixing. Much less mixing occurs in the swap protocol that in the protocol.

2.3.1 Swap Protocol

The swap protocol can be viewed as a stripped-down version of the previous protocol because nodes only return one edge instead of their whole views when contacted. Alternatively, one can think of the swap protocol as an edge swapping protocol: two nodes exchanging an edge.

Each node A repeatedly initiates the following:

1. Node A contacts a random node B chosen from its local view
2. Node B then selects a random element C from its view and returns it to A
3. Node A replaces element B by C in its view
4. Node B replaces element C by A in its view.

See Figure 2.3 for a graphical representation.

Node A has a pointer to B, and B has a pointer to C. Node A contacts B which return its pointer to C. Then A replaces its edge to B by an edge to C (mixing), B drops its edge to C and creates an edge to A (reinforcement).

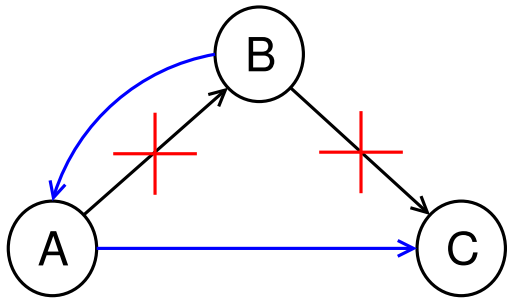


Figure 2.3: Graphical Representation of the Swap Protocol

2.3.2 Mixing and Reinforcement

At each update, one edge is mixed and one node is reinforced. Compare this to the previous protocol where a whole view, that is, k edges were mixed per f reinforcements. In another words, the swap protocol is such that the amount of reinforcement is much higher than in the original protocol. It appears that a protocol delivering more reinforcement than the swap protocol cannot exist.

Note on Edge Liveness The edge mixing is very slow in the swap protocol. Contrary to other gossip-based membership protocols, a node is quite likely to actually communicate with the end point of any of its edges. There is a case for monitoring the liveness of the edge targets, thus purging dead links quicker. We chose not to investigate.

Checking for the liveness of the edges in one's view is a complete waste of time in most of the other gossip-based membership algorithms from a node point of view. Chances are that it will never attempt to use that edge.

2.3.3 Non-Partitioning

The swap algorithm guarantees that the graph always stays connected if no node fails. Consider a path $\mathcal{P}$ connecting two nodes X and Y. The only edges changing during one iteration of the protocol are the two edges A to B and B to C.

1. The edge between A and B is flipped. There is no change for any path using that edge (remember that the graph is considered undirected for non-partitioning purposes).

2. The edge between B and C is replaced by an edge between A and C. Consider any path using the edge B to C and replace that edge by two edges, one from B to A and one from A to C which both exist because the protocol just created them.

This proves that any pair of connected nodes stays connected after one iteration of the protocol. Thus they will stay connected after any number of iterations.

2.4 Alternative Protocols

There were several choices made in the design of the main algorithm: push, pull, randomized, etc. Here we explore some of the alternatives and cite some relevant work. Reference [41] simulates many of these alternatives. Note however that this reference does not distinguish between reinforcement and pulling (view mixing), using the same process for both. We point out their results where applicable.

Scamp [29] and follow up papers [28] explore mechanisms to dynamically adjust the view size to the logarithm of the number of network nodes. As we already mentioned (in Section 2.1.3) automatically adjusting the view size is outside the scope of the present work.

2.4.1 Reinforcement

Reinforcement is implemented using a push mechanism: every node pushes its name onto the view of some (random) node. Consider the array $\mathcal{V}$ defined by the concatenation of the node views. The number of times a node u appears in the array is equal to node u's in-degree and corresponds to the number of nodes having u in their views. Node u may be dropped from the concatenation $\mathcal{V}$ of the views due to selection randomness in the pulling part at each protocol iteration or to nodes failing, leading their views to disappear from the system. The idea of reinforcement is to counterbalance this potential drop by having nodes actively add their name to the concatenation $\mathcal{V}$ of the views. This analysis will be detailed in Chapter 5.

The alternative implementation of reinforcement, a pull mechanism, makes little sense. Adding node u to one's view is only useful if u is not already present in the view. In a pull mechanism, the nodes that do not have node u in their views cannot contact u, thus will never reinforce u. The nodes that do have u in their view have no need to reinforce u. In a pull mechanism, no reinforcement happens. As we shall prove in Chapter 5, lack of reinforcement yields a star network. This has also been noted in [41].

2.4.2 Mixing

The mixing part of the protocol was implemented by a pull mechanism for simplicity. Really, it is about mixing the content of the views. Pushing, pulling or push-pull are valid options. However, we chose pulling and have the following non-rigorous argument to support our choice. Set reinforcement aside and consider the (directed) connectivity graph $\mathcal{G}$ at $t = 0$. Under a pull mechanism, an element of

the view of node i at time t is obtained by a random walk of 2^t steps in $\mathcal{G}$ starting from i. This might suggests the following memoryless property: after $\mathcal{O}(\ln n)$ rounds, the views are independent of their initial value.[2] This will be explained in more details in Chapter 7.

Under a push mechanism, the random walk in not necessarily directed and can be significantly shorter since the walk can actually backtrack. Reference [41] also notes that Push increases the risk of partitioning when the network grows.

When implementing view mixing and reinforcement altogether like in Cyclon [80], push-pull ensures the presence of the good features of both the push and pull mechanisms.

2.4.3 Randomization

The random choices the protocol makes do not have to be uniform. For example, each view entry could have a time-stamp. Then the random choices (communication partner, replacement) can be based on these time-stamps, *e.g.* selecting to communicate with the node with the highest time-stamp. Again, see [41] for some simulations.

In our protocol, a complex randomized process ensures that at each iteration some edges are dropped from the set of the views and replaced by fresh edges. The variance in the lifespan of the edges is the primary source of node in-degree variance. Time-stamps can advantageously replace randomization, sharply decreasing

[2]Developing this argument suggests modifying the protocol to update only a randomly chosen half of the view at each update so as to make the random walk lazy, ignoring the fact that the walks for different elements are correlated, and a liberal application of the results of [14]. Fan Chung proves that lazy random walk on directed regular graphs mix in small polynomial time. Here our graph is close to regular.

the in-degree variance. See Cyclon [80] for a protocol quite similar to ours, but using time-stamps.

Chapter 3

Dynamic Behavior of the Protocol

In this chapter we present a a simple yet powerful framework to analyze the behavior of the protocol. We consider a partition A-B of the network graph. The fraction of edges pointing to nodes in A is an estimate of the fraction of A nodes. We consider two estimates, the one made by the A nodes and the one made by the B nodes. We study the evolution of these estimates. This is sufficient to predict the overall behavior of the membership algorithm. In particular the analysis shows that any significant lack of edges across a given network cut will be corrected extremely rapidly. The analysis also shows that the network diameter is small.

3.1 Definitions, Partitioning and Size Estimates

3.1.1 Definitions

Consider a partition of the set of nodes into two sets A and B. Let x be the fraction of edges from nodes in A that go to nodes in A, as illustrated by the following formula:

$$x = \frac{1}{\text{View size} \times |A|} \sum_{\text{node} \in A} \#(\text{edges to nodes in A})$$

Conversely let y be the fraction of edges from nodes in B that go to nodes in A:

$$y = \frac{1}{\text{View size} \times |B|} \sum_{\text{node} \in B} \#(\text{edges to nodes in A})$$

For our purposes, the graph is partitioned if and only if there are no edges across the partition A-B, that is, both $x = 1$ and $y = 0$.

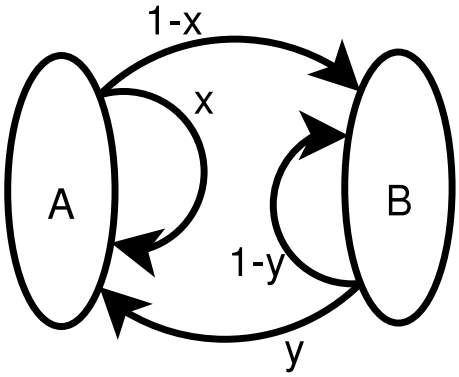


Figure 3.1: Edge Fractions

We shall see later on that we always have $x \approx y$, hence there are edges across the A-B cut. Applying this fact to all possible A-B partitions, we see that the graph cannot be partitioned.

3.1.2 Estimate of Size

We revisit the above definition of x. Note that if the edges were drawn uniformly at random, the expected number of edges to nodes in A (from nodes in either A or B) would be proportional to the fraction of A nodes.

Consider a set S of nodes. If these nodes think A represents 30 % of the nodes, and B the other 70 %, one expects 30 % of the edges from nodes in S to point to A, and 70 % to point to B, even if A is only 10 % of the nodes. The fraction of edges pointing to A is the estimate of the normalized size[1] of A by the nodes of S. Applying this to set A instead of S, we see that x is the estimate by the A nodes of the size of A. Conversely, y is the estimate by the B nodes of the size of A.

We saw earlier that the graph is partitioned if and only if $x = 1$ and $y = 0$.

[1] number of nodes in the considered set divided by the total number of nodes

The graph is far from being partitioned if both parts agree on their estimate of the size of A: $x = y$, and this is what the protocol does: it ensures that both sides agree on their estimates by constantly averaging them.

The estimate may be very far from the correct size. This has no bearing on the partitioning issue: the network is unbalanced but not partitioned. Consider an extreme example: partition the graph evenly into two parts A and B, and let $x = y = 1$. All the nodes think that B does not exist. But B is not partitioned from A: even though there are no edges from A to B, all edges from B go to A. In fact, half the edges in the graph are edges from B to A.

3.2 Evolution of the Size Estimates

We now describe the evolution of the estimates of the size of A. Note the slight abuse of notation. When referring to the size of A, we refer to the normalized size of A, that is, the fraction of A nodes.

3.2.1 Evolution of the Size Estimate in Our Protocol

Mixing

Neglecting reinforcement, it is easy to see that both A and B converge to the same estimate of the size of A. The act of pulling and merging the views corresponds to asking nodes from both sides for their estimate, then averaging them.

Let x_t and y_t denote the fractions of edges to set A after t iterations of the protocol. Considering the A nodes, at time $t + 1$ with probability x_t they pull nodes from set A (and get a fraction x_t of edges to A), and with probability $1 - x_t$ they pull nodes from B (and get a fraction y_t of edges to A). Similarly for y_t. Hence

we get:

$$\begin{cases} x_{t+1} &= x_t * x_t + (1 - x_t) * y_t \\ y_{t+1} &= y_t * x_t + (1 - y_t) * y_t \end{cases} \tag{3.1}$$

Both new estimates of the size of A, that is, the estimate x_{t+1} from the A nodes, and the estimate y_{t+1} from the B nodes, are a weighted average between A's and B's current estimate, the weight being proportional to the estimated size of A.

This system of equations defines a recurrence, which convergences very rapidly to $x = y$ as long as the initial values are not $x_0 = 1$ and $y_0 = 0$, or $x_0 = 0$ and $y_0 = 1$. The former corresponds to a partitioned graph, the latter to a graph where all edges cross the partition. The latter gives a partitioned graph in one iteration, but only because we neglect reinforcement. All other initial values converge very rapidly to $x = y$.

Indeed we have

$$x_{t+1} - y_{t+1} = (x_t - y_t)(x_t - y_t) = (x_t - y_t)^2$$

which means that, starting from x_0 and y_0, in t iterations, we get

$$(x_t - y_t) = (x_0 - y_0)^{2^t} \tag{3.2}$$

which converges to $x_t = y_t$ as long as we don't have both $x_0 = 1$ and $y_0 = 0$, or $x_0 = 0$ and $y_0 = 1$.

The estimate of the size of the set A by the A nodes, and that same estimate by the B nodes, quickly converge to the same value. This value, however, has no tangible reason for being the true size of A. Agreeing on the estimate ensures that the graph is not partitioned. It also ensures some conditions that are necessary for the reinforcement to perform as expected, which then brings the estimate to the correct value. This will be detailed in Section 3.3 on page 35.

Reinforcement

Consider set A, with both sides agreeing on their estimate of the size of A. Some nodes from A and some nodes from B are going to pull A nodes. But what proportion of each? Actually, the same proportion as the sizes of A and B. Say $x = y = 30\,\%$ to fix the ideas. $30\,\%$ of the A nodes pull from A and $30\,\%$ of the B nodes pull from A. Imagine the entity "set A" is a restaurant. Its clients are the nodes that pull from set A, that is, $30\,\%$ of the A nodes and $30\,\%$ of the B nodes. From the restaurant's point of view, its clients formed a representative sample of the whole population. The ratio of A clients to B clients at Restaurant "Set A" is the same as the ratio of A nodes to B nodes in the overall population.

This is why the reinforcement brings the estimate of the size to the correct value: it injects a little bit of the true value at each iteration. Note, this only works when both sides agree on their estimate of the size of A. Otherwise, A would not be pulled by the correct proportion of nodes. Note also that by symmetry, the same applies to B. This will be detailed in Section 3.3 on page 35.

3.2.2 Diameter

Agreement on the sizes ensures that the diameter of the graph is small. Assume $|A| = \gamma n \leq |B|$ and consider the number of edges across the cut. There are $(1-x)\gamma kn$ edges from A to B and $y(1-\gamma)kn$ edges from B to A. Since $x = y$ there are $(1-x)\gamma kn + x(1-\gamma)$ edges across the cut. We chose $\gamma \leq 1/2$, that is, $(1-\gamma) \geq \gamma$ so the number of edges across the cut is at least:

$$(1-x)\gamma kn + x(1-\gamma)kn \leq (1-x)\gamma kn + x\gamma kn = \gamma kn$$

This property makes the undirected graph an expander graph[2].

An application-level push-pull gossip broadcast will converge rapidly on such a graph. [23, 64] Many other gossip algorithms will also converge rapidly. However, some care needs to be taken with applications that rely on the undirectedness of the underlying graph. Consider the typical case of a node X having an edge to Y and node Y not having an edge to X. While nodes X and Y can exchange any information once the communication has been initiated, node Y needs to wait (or hope) for X to initiate the communication and cannot initiate on its own. This is a break of symmetry compared to the undirected case. Nevertheless, we are not aware of gossip applications for which this is an issue.

3.2.3 On the Feasibility of a Result with High Probability regarding the Evolution of the Size Estimates

In the previous section we derived a result on the size estimates. This result was in expected value. Obtaining a result which holds with high probability appears to be very difficult. While it is quite simple to obtain a similar result to Equation (3.2) on page 31 to hold with very high probability for a given partition of the nodes, this is not sufficient.

Indeed, we are not only concerned about these two sets separating, being left with no cross edges. We are worried about this happening for *any* partition of the nodes into two sets. If the result with high probability for a given partition was strong enough to sustain a union bound on all 2^n partitions, we would be set. However, standard techniques like Chernoff-Höeffding bounds will not yield a good

[2] Roughly defined as k-regular undirected graphs with the property that any small subset of vertices has a relatively large neighborhood [13, 88].

enough result.

Taking a union bound on all 2^n partitions might be overkill. There might be a way to group together various partitions of the set of nodes to significantly decrease the number of objects on which the union bound is taken. This approach is certainly promising and is worth pursuing further.

3.2.4 Evolution of the Size Estimate in Other Protocols

Equations 3.1 on page 31 are specific to our protocol. However, similar equations could easily be written for other protocols. For example, we considered a push only protocol, that is, instead of contacting nodes to retrieve their views, each node contacts a given number of nodes to whom it pushes its own view. This corresponds to a simplified version of LwPBCast [22]. The equations are the following:

$$\begin{cases} x_{t+1} &= \frac{x_t * x_t + y_t * y_t}{x_t + y_t} \\ y_{t+1} &= \frac{(1-y_t)^2 + (1-*x_t)^2}{(1-y_t) + (1-*x_t)} \end{cases} \tag{3.3}$$

By defining $z_t = 1 - y_t$, the fraction of B edges pointing to B, the equations become symmetric in x_t and z_t. These equations are more complex than that of the pull model (3.1) on page 31 and cannot be analyzed analytically. However, numerical simulations gave the same result as Equation (3.2) on page 31. We conjecture that the same result will also be obtained in the case of a push-pull algorithm.

Swap-Algorithm Here again, the equations do not allow for analytical study. Probabilistic equations can be written for a single update instead of per round as done previously. Then, numerical simulations show that the size estimates converge to agreement, though at a slower rate than with the other protocols.

This is not surprising since the amount of mixing made by this algorithm per round is significantly less.

3.3 Equations in Expected Sense

We now consider both the mixing and the reinforcement. We shall write equations capturing the overall behavior. First, let us define γ to be the fraction of A nodes:

$$\gamma = \frac{|A|}{|A| + |B|}$$

We start by looking at the totality of the edges the A nodes consider for selection for their next view. Our aim is to calculate the expected value for one node, but it is easy to consider all the nodes from A at once, and later divide by the number of A nodes. These edges considered for selection come either from the mixing (pulling from other nodes), or the reinforcement. The breakdown is as follow:

Mixing The total is $|A|fk$, where f is the fanout, is the number of views a node requests during one iteration of the protocol. This can be split into two parts: edges which point to A, and edges which point to B.

How do we get an edge from A to A? Considering an A node: We either pull from a node in A (with expected probability x), and that node has an expected fraction x of its k edges pointing to A; or we select a node from B (with expected probability $1 - x$), which has an expected yk edges to A.

How do we get an edge from A to B? We either pull from a node in A (with probability expected x), and that node has an expected $(1 - x)k$ edges pointing to B; or we select a node from B (with expected probability $1 - x$), which has a expected $k(1-y)$ edges to B. We can do the same for part B. We get, with $X \to Y$

denoting edges from X to Y:

$$\begin{aligned}
A \to A: &\quad |A|fk[x^2 + (1-x)y] \\
A \to B: &\quad |A|fk[x(1-x) + (1-x)(1-y)] = |A|fk(1-x)(1+x-y) \\
B \to A: &\quad |B|fk[(1-y)y + yx] = |B|fky(1+x-y) \\
B \to B: &\quad |B|fk(1-y(1+x-y)) = |B|fk[(1-y)^2 + y(1-x)]
\end{aligned}$$

Reinforcement We shall enumerate all the cases. Each time somebody pulls, it leaves its name exactly once for reinforcement, and each node pulls f times. $X \leftarrow Y$ means X pulls from Y, and the edge created is from Y to X.

$$\begin{aligned}
A \leftarrow A: &\quad f|A|x \\
A \leftarrow B: &\quad f|A|(1-x) \\
B \leftarrow A: &\quad f|B|y \\
B \leftarrow B: &\quad f|B|(1-y)
\end{aligned}$$

The sum is of course $f(|A| + |B|)$

Combining Both The above numbers are for the whole set, we need to divide by $|A|$ for expected value for an A node, and by $|B|$ for a B node.

The number of edges pointing to A from an A node after one iteration, denoted by nb, is:

$$nb = \frac{\text{Edges to } A \text{ from Mixing} + \omega \times \text{Edges to } A \text{ from Reinforcement}}{\text{Total Number of edges mixed} + \omega \times \text{Total Reinforcement}}$$

Remember ω is the function from the definition of reinforcement in the protocol (see Section 2.2.1 on page 16) that represents the weight given to reinforcement.

We now want to take the expected value of the above fraction, and plug in the values we got above, to get:

$$E[nb] = \frac{\frac{|A|fk[x^2+(1-x)y]}{|A|} + \omega\frac{f|A|x}{|A|}}{fk + \omega\frac{f|A|x}{|A|} + \omega\frac{f|B|y}{|A|}}$$

This means writing

$$E[nb] = E[\frac{a+b}{c+d}] = \frac{E[a+b]}{E[c+d]} = \frac{E[a]+E[b]}{E[c]+E[d]} = \frac{\frac{|A|fk[x^2+(1-x)y]}{|A|} + \omega\frac{f|A|x}{|A|}}{fk + \omega\frac{f|A|x}{|A|} + \omega\frac{f|B|y}{|A|}}$$

Going from *right to left* in the previous series of equalities: a is the product of independent variables (the fraction of edges from the current node to A or B, and the number of A edges that pulled the current node), so we have $E[a] = \frac{|A|fk[x^2+(1-x)y]}{|A|}$. The term c is a constant (k, the view size). There is independence as well between the considered node and the nodes that pull it, hence $E[b]$ and $E[d]$. Left is the fraction issue: we need $E[x/y] = E[x]/E[y]$ (with $x = a+b$ and $y = c+d$), which in general is untrue. Here however this is approximatively true, because the variable term in the denominator $E[d]$ is small compared to constant $E[c]$.

Finally we get:

$$\begin{aligned} E[nb] &= \frac{\frac{|A|fk[x^2+(1-x)y]}{|A|} + \omega\frac{f|A|x}{|A|}}{fk + \omega\frac{f|A|x}{|A|} + \omega\frac{f|B|y}{|A|}} \\ &= \frac{\gamma k[x^2+(1-x)y] + \omega\gamma x}{\gamma k + \omega\gamma x + \omega(1-\gamma)y} \end{aligned}$$

We went from the first line to the second one by multiplying each element by $|A|/(|A|+|B|) = \gamma$.

This expected number is actually the new value of x. We can do the same for y, and we get:

$$\begin{cases} x_{n+1} &= \frac{\gamma k[{x_n}^2+(1-x_n)y_n]+\omega\gamma x_n}{\gamma k+\omega\gamma x_n+\omega(1-\gamma)y_n} \\ y_{n+1} &= \frac{(1-\gamma)k[y_n(1+x_n-y_n)]+\omega\gamma(1-x_n)}{(1-\gamma)k+\omega\gamma(1-x_n)+\omega(1-\gamma)(1-y_n)} \end{cases} \tag{3.4}$$

The first order term in k of Equation (3.4), yields exactly the Equations (3.1) on page 31 developed while neglecting the reinforcement. This validates our hypothesis of neglecting the reinforcement in the first approach.

The only fixed points in Equations (3.4) on the preceding page are:

1. $\begin{cases} x = 1 \\ y = 0 \end{cases}$ the graph is partitioned.

2. $\begin{cases} x = \gamma \\ y = \gamma \end{cases}$ equilibrium, as if the graph was drawn uniformly at random.

Ignoring the partitioned case, notice that by considering the reinforcement, there is now one single equilibrium point. Having $x = y$ is not sufficient to have equilibrium anymore, unless $x = y = \gamma$. This is exactly what we wanted: that x and y – the estimates of the size of A – converge to the true size of A.

Uninteresting algebra proves the following behavior, as illustrated by a few simulation runs on Figure 3.2 on the following page. We do not consider the initial configuration $x = 1$, $y = 0$ corresponding to the partitioned case in our discussion.

1. All starting configurations converge to γ, γ, the correct estimate.

2. We always have $x \geq \gamma$ and $y \leq \gamma$ (except maybe for the initial configuration). This means that each side always overestimates its own size and underestimates the size of the opposite side.

3. The convergence to the line $x = y$ is extremely fast. Note that it is not exactly a line $x = y$, more a curve on which $x \approx y$. It is a line in the limit as k goes to infinity.

4. Once on the $x = y$ curve, the reinforcement kicks in, and drags the estimates to the correct value.

5. Before being on the $x = y$ curve, the reinforcement does not have much of an effect and does not even necessarily push in the correct direction.

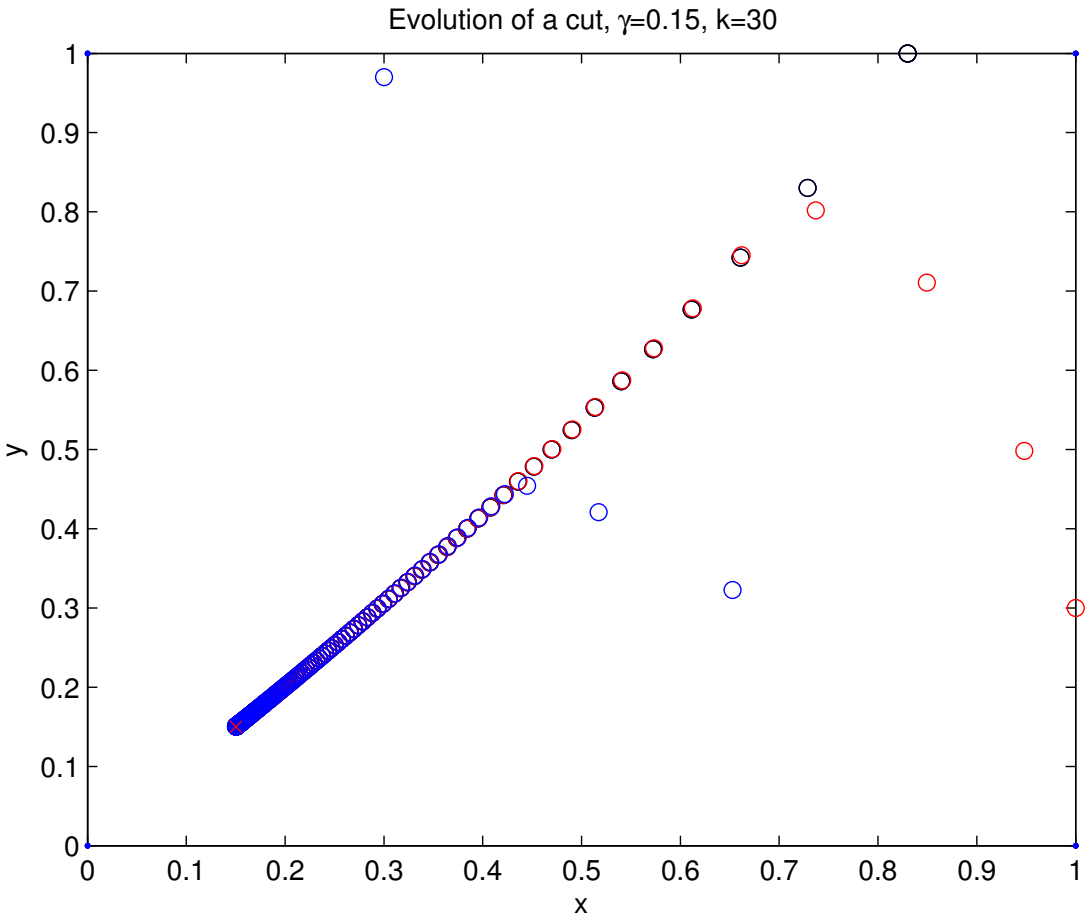


Figure 3.2: Three Different Runs of the Convergence

6. The reinforcement does not maintain the $x = y$ invariant. This is because when x and y are different from γ, both values are pushed towards γ, though not by the same amount. However, the mixing brings x and y back to an agreement.

   This actually explains why the curve $x = y$ is not exactly the line $x = y$, but a curve where $x \approx y$.

Chapter 4

Non-Partitioning

In this chapter we show that the expected time before the network partitions is exponentially small in both the size of the departing partition and the size of the views. In other words, for a sufficiently large system with moderately large views, the system will *never* partition: the only disconnected components, if any, will be lone nodes whose view edges are all dangling.

We first introduce a framework capturing the essence of the main protocol. A few minor modifications would be needed to capture the swap protocol. The simplifications made in deriving the framework allow us to write equations capturing the evolution of the system and derive the expected time before a set of nodes becomes disconnected from the rest of the network.

Our main result is the following. Denote by s the size of the departing set of nodes and k the view size. Let μ be the churn rate: at each round of the protocol, μ randomly selected nodes die, and μ new nodes join the network. The expected time until a set of $s \leq n/2$ nodes parts away from the rest of the network is exponentially large in both s and k. Our proof holds as long as $\mu \ll k$.

4.1 Model

Nodes are not updated all at once. Instead, a single node updates a single entry in its view at a time. More exactly, at each iteration of the modified protocol, one node, chosen uniformly at random, updates one element chosen uniformly at random from its view. That is, at each iteration in our modified protocol, when pulling, a node u chosen uniformly at random replaces a (randomly chosen) node v

in its view by a (randomly chosen) node from the view of node v. Node u updates a single element of its view instead of all of them. Reinforcement works as follows: at certain times, a node u chosen uniformly at random tags a randomly chosen node v from its view. Then node v reinforces u by replacing one of the nodes in its view by u.

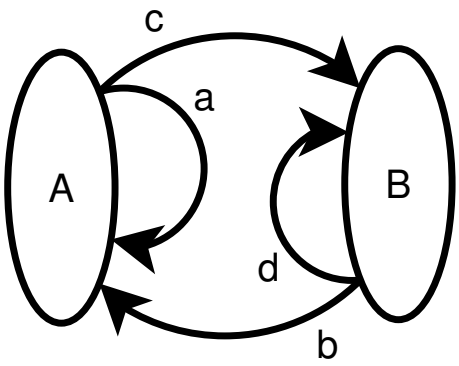
4.1.1 Assumption

Our assumption is the following: consider a partition of the set of nodes into two sets A and B. Each node has some number of edges pointing to nodes in A, and some number of edges pointing to nodes in B. We assume a node's edge distribution to be independent of that of its neighbors: Consider a node u and some neighbor $v \in$A of node u. The expected number of edges node v has pointing to set A is independent of the fact that node v is a neighbor of node u. Any other node $x \in$A has in expected value the same number of edges pointing to set A.

In other words, the (out-)edges of A nodes are identically distributed. The same holds for B nodes. The need for this assumption is clear when looking at Equations (4.1) on page 47. Consider a node u having an edge to a node $v \in$A, the assumption allows us to consider a random node $x \in$A instead of node v. While this is clearly incorrect for a system with a small number of nodes, we believe the approximation to hold when the number of nodes is large.

4.1.2 Model

Consider a partition of the set of nodes into two sets A and B. Let γ be the fraction of A nodes and let $\mathcal{A}$ and $\mathcal{B}$ be two arrays. Consider the views of the A nodes. For each occurrence of an A node, place a 0 in the array $\mathcal{A}$, and for each occurrence



$$a + c = \gamma kn$$

$$b + d = (1 - \gamma)kn$$

Figure 4.1: Number of Edges

of a B node, place a 1 in the array $\mathcal{A}$. Let a be the number of 0's (the number of edges from nodes in A to nodes in A) and c (as in **c**ross edges) the number of 1's (the number of edges to nodes in B). The length of $\mathcal{A}$ is $a + c = \gamma kn$. Recall $\gamma = {}^{|A|}/_{n}$.

Conversely, consider the views of the B nodes. Place a 0 in the array $\mathcal{B}$ for each occurrence of an A node, and place a 1 for each occurrence of a B node. Let b be the number of 0's, and d be the number of 1's. There are $b + d = (1 - \gamma)kn$ elements in $\mathcal{B}$. Each array represents the concatenation of the views of the nodes of one set where a 0 denotes an edge pointing to a node in A, and a 1 denotes an edge pointing to a node in B.

Mixing (pull) updates can now be made precise. If the element being updated is a 0, the new value is the value of a position chosen uniformly at random from $\mathcal{A}$. If it is a 1, the value is the value of a position chosen uniformly at random from $\mathcal{B}$. As for reinforcement, if the node doing the tagging is in A, put a zero, otherwise, a one.

The modifications we made to the protocol ensure that a single array entry is updated each time. To correctly mimic the protocol, with probability p the update will be reinforcement, and with probability $q = 1 - p \gg p$ the update will be by pulling, where the parameter p is an increasing function of the protocol parameter "Weight of Reinforcement" ω, taking value 0 when $\omega = 0$, $1/k$ when $\omega = 1$, and f/k when $\omega = \infty$. In a later claim, we require $p \ll 1/\sqrt{\ln \gamma kn}$. This inequality is verified even when $p = f/k$, for all values of γ by having $f \lessapprox \sqrt{k}$. This is the case since we take f to be a small constant.

As we have seen in the previous chapter, we expect the fraction of zeros in both sides, that is, $a/\gamma kn$ and $b/(1-\gamma)kn$, to converge to the same value. If and and only if the system is partitioned do we have $a = \gamma kn$ and $b = 0$, equivalent to $b = c = 0$.

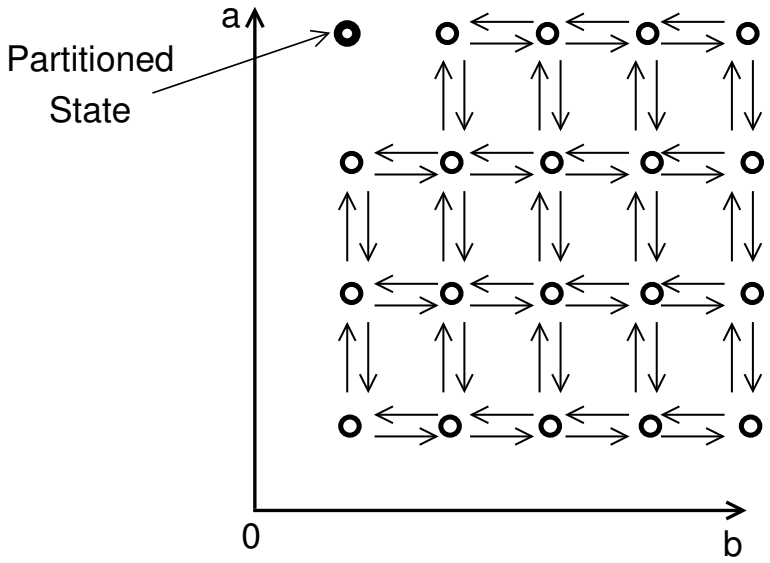
For simplicity we assume the churn rate μ to be zero. The algebra details for $\mu \neq 0$ will be explained in Section 4.4.

4.1.3 State Diagram

Our model boils down to studying the evolution of these four values: a, b, c and d. In fact, the two values a and b fully characterize the problem. So do b and c. This corresponds to a discrete Markov Chain governed by equations which will be described in Section 4.2 below. The state diagram is a rectangular grid, with $0 \leq a, c \leq \gamma kn$ and $0 \leq b, d \leq (1-\gamma)kn$. Using our modified protocol, the only possible transitions in the state diagram are to move one position to the left, to the right, up or down, subject to not stepping out of the grid.

To make our discussion easier, let us agree that on the grid (Figure 4.2 on the following page) the vertical axis is for a and c, with a increasing and c decreasing going from bottom to top on the grid. On the horizontal axis, b is increasing and

d is decreasing going from left to right. The state corresponding to a partition is given by $a = \gamma kn$, $b = 0$.



a is the number of edges from nodes in A to nodes in A

b is the number of edges from nodes in B to nodes in A

Figure 4.2: State Diagram, no reinforcement

Reinforcement

Without reinforcement, the results from Section 5.2 on page 73 apply: in $\mathcal{O}(kn \ln kn)$ steps, the system converges to both arrays being all zeros or all ones. This situation denotes a state in which either all the edges from A and B point to nodes in A or all the edges from A and B point to nodes in B. The system is stable and unpartitioned. However, this is not really a satisfactory result: the system does not partition because it is stuck in a "dead-state."

Without reinforcement, a few transitions are missing in the state diagram (See Figure 4.2). Most importantly, there are no transitions into the partitioned state.

Also, there are no transitions out of any state having $a = \gamma kn$ or $b = 0$ (any state where all edges from one set point to the other). Adding reinforcement adds the missing state transitions and ensures that the system does not get stuck in any state except the partitioned state ($a = \gamma kn$, $b = 0$). This almost yields the state diagram represented on Figure 4.3 on the following page.

Limit Probability Distribution

As it stands, the chain is connected and there is a single sink (dead) state, the partitioned state. The limit probability distribution is 1 for the partitioned state ($a = \gamma kn$, $b = 0$), and 0 for the rest of the grid, because there is no transition out of the partitioned state. However, it takes a long time for the probability distribution to converge to the limit distribution. For all practical considerations, a limit distribution achieved only after all processors have rusted and the universe has died is of little interest.

We modify the Markov Chain by adding a transition out of the partitioned state ($a = \gamma kn$, $b = 0$) into the two neighbor states ($a = \gamma kn - 1$, $b = 0$) and ($a = \gamma kn$, $b = 1$). See Figure 4.3 on the following page for the state transitions of the modified Markov Chain. Each time the graph partitions, we restart with a single directed edge across the split. The fraction of time spent in the partitioned state in this never-ending process is a lower bound on the expected time until the (real) system partitions.

Claim

In the model described above, the fraction of time spent in the partitioned state of this finite (2-dimensional) Markov Chain is exponentially small in γkn when

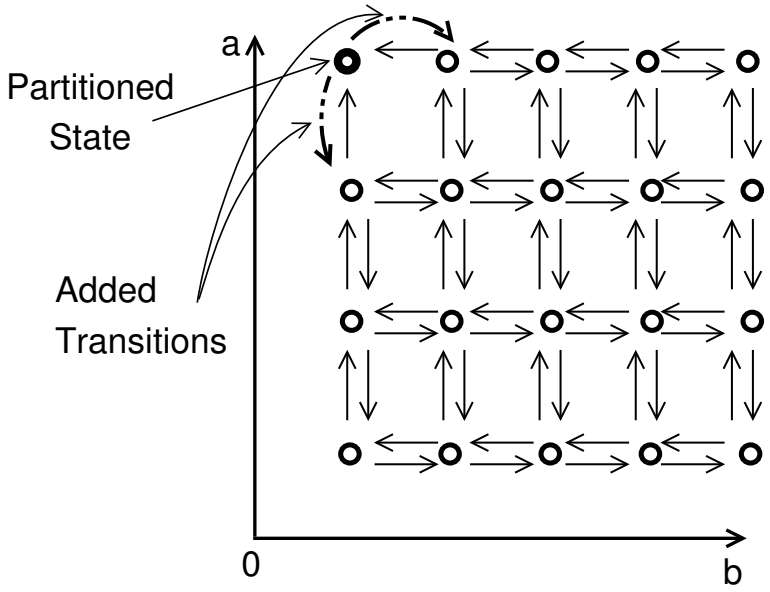


Figure 4.3: State Diagram, with reinforcement and added transitions out of the partitioned state

$\mu \ll \gamma kn$ and $p \ll 1/\sqrt{\ln \gamma kn}$.

The above result on the time to partition proves that if, and when, the network partitions, only a very small component disconnects, and the rest of the network stays connected. Furthermore, the nodes of the small disconnected component can easily detect they have partitioned away by looking at the (lack of) diversity in the content of their views over time. They then attempt to re-join the network. Setting $\gamma = 1/n$ in the above formula means considering a single node and its probability of getting disconnected for having only dangling edges in its view. The view size needs to be larger than the churn rate for the expected time until partition to be exponential.

4.2 Equations

Let p be the probability of reinforcement. The probability of a pull update is $1-p$. The reader might want to refer to Equations (3.2.1) on page 32 for the reinforcement part of the following equations.

The probability of changing the number of zeros in $\mathcal{A}$ is:

$$\begin{aligned}
\Pr[(a,b) \to (a-1,b)] &= (1-p)\frac{a}{kn}\frac{c}{\gamma kn} + p\frac{b}{kn}\frac{a}{\gamma kn} \\
&= \frac{a}{k^2n^2}\left(\frac{(1-p)c}{\gamma} + \frac{pb}{\gamma}\right) \\
\Pr[(a,b) \to (a+1,b)] &= (1-p)\frac{c}{kn}\frac{b}{(1-\gamma)kn} + p\frac{c}{kn}\frac{a}{\gamma kn} \\
&= \frac{c}{k^2n^2}\left(\frac{(1-p)b}{1-\gamma} + \frac{pa}{\gamma}\right)
\end{aligned} \tag{4.1}$$

The first term on the right-hand side of the first equation corresponds to updating by pulling. To decrease the number of 0's in the array $\mathcal{A}$, select for replacement a 0 in $\mathcal{A}$, which happens with probability a/kn. This 0 denotes an edge pointing to a node in A. We follow that edge and use the target's value for our update. In our model, "following the edge" means "choosing a position in $\mathcal{A}$ uniformly at random."[1] We need to pick a one in $\mathcal{A}$, which happens with probability $c/\gamma kn$.

Reinforcement occurs with probability p. For the number of zeros in $\mathcal{A}$ to decrease, a node from B must tag a node from A, which happens with probability b/kn, and the element in $\mathcal{A}$ receiving the reinforcement needs to be a zero, which happens with probability $a/\gamma kn$. Hence:

$$\Pr[(a,b) \to (a-1,b)] = (1-p)\frac{a}{kn}\frac{c}{\gamma kn} + p\frac{b}{kn}\frac{a}{\gamma kn}$$

A similar reasoning explains the increase in the number of zeros by 1 in $\mathcal{A}$; hence the other part of Equations (4.1). We shall not present the analysis for the

[1] This is where our assumption about neighbor independence is being used.

evolution of $\mathcal{B}$, since it can be obtained by symmetry, exchanging a by d, b by c, and γ by $1-\gamma$. The probabilities of changing the number of zeros in $\mathcal{B}$ are:

$$\begin{aligned}
\Pr[(a,b)\to(a,b-1)] &= (1-p)\frac{d}{kn}\frac{b}{(1-\gamma)kn}+p\frac{c}{kn}\frac{d}{(1-\gamma)kn} \\
&= \frac{d}{k^2n^2}\left(\frac{(1-p)b}{(1-\gamma)}+\frac{pc}{(1-\gamma)}\right) \\
\Pr[(a,b)\to(a,b-1)] &= (1-p)\frac{b}{kn}\frac{c}{\gamma kn}+p\frac{b}{kn}\frac{d}{(1-\gamma)kn} \\
&= \frac{b}{k^2n^2}\left(\frac{(1-p)c}{\gamma}+\frac{pd}{1-\gamma}\right)
\end{aligned} \tag{4.2}$$

Neglecting reinforcement in these equations, the probability to step towards the diagonal line $a/\gamma = b/(1-\gamma)$ (corresponding to both sides agreeing on their estimate of the size of A), is larger than stepping away. In other words, the system has a natural tendency to re-equilibrate itself.

4.3 Proof

We shall prove that the limit probability of the partitioned state $a=\gamma kn$, $b=0$ is exponentially small in γkn. We consider the steady state. As such, all probabilities are understood as steady state probabilities, that is, the value in the so called limit probability distribution. Accordingly, $P_{(a,b)}$ denotes the (steady state) probability of being in state (a,b).

The idea behind the proof is the following: assume that

$$\begin{cases}
P_{(a,b)}\Pr[(a,b)\to(a+1,b)] &= P_{(a+1,b)}\Pr[(a+1,b)\to(a,b)] \\
P_{(a,b)}\Pr[(a,b)\to(a-1,b)] &= P_{(a-1,b)}\Pr[(a-1,b)\to(a,b)] \\
P_{(a,b)}\Pr[(a,b)\to(a,b+1)] &= P_{(a,b-1)}\Pr[(a,b+1)\to(a,b)] \\
P_{(a,b)}\Pr[(a,b)\to(a,b-1)] &= P_{(a,b-1)}\Pr[(a,b+1)\to(a,b)]
\end{cases} \tag{4.3}$$

holds for all a's and b's. Consider a cut. At equilibrium, the amount of probability flow crossing the cut in one direction equals the amount of flow crossing the cut in

the opposite direction. Here we assume this to be the case for any individual edge, not just on average on the edges defining a cut. While clearly if Equations (4.3) were to hold we would have equilibrium, the converse is not true. Equilibrium does not imply that Equations (4.3) are true. In fact, these equations are so numerous that taken altogether, they do not have a solution.

Assuming there is equilibrium along any edge, we then have

$$\begin{aligned}
\frac{\Pr[(a,b)\to(a+1,b)]}{\Pr[(a+1,b)\to(a,b)]} &= \frac{P_{a+1,b}}{P_{a,b}} \approx \frac{\gamma}{1-\gamma}\frac{b}{a} \\
\frac{\Pr[(a,b)\to(a,b-1)]}{\Pr[(a,b-1)\to(a,b)]} &= \frac{P_{a,b-1}}{P_{a,b}} \approx \frac{(1-\gamma)}{\gamma}\frac{(\gamma kn-a)}{b-(1-\gamma)kn} = \frac{(1-\gamma)}{\gamma}\frac{c}{d}
\end{aligned} \tag{4.4}$$

The simplification comes from dropping the terms corresponding to reinforcement (the terms in p) in Equations (4.1) on page 47 and (4.2) on the preceding page. The remaining two terms are smaller than one for any position above the diagonal: $a/\gamma n \geq b/(1-\gamma)n$, and decreasing as we step away from the diagonal and get closer to the partitioned (dead) state.

Consider any path between a state on the diagonal and the partitioned state such that for each transition, either a increases, or b decreases, that is, we only use the two transitions written in Equation (4.4). Express the ratio between the limit probability of these two states as the cascading product of the limit probability distribution of the intermediary points on the path:

$$\begin{aligned}
\mathcal{R} &= \frac{P_{\gamma kn,0}}{P_{a_{\text{start}},b_{\text{start}}}} \\
&= \frac{P_{a_1,b_1}}{P_{a_{\text{start}},b_{\text{start}}}}\frac{P_{a_2,b_2}}{P_{a_1,b_1}}\frac{P_{a_3,b_3}}{P_{a_2,b_2}}\cdots\frac{P_{\gamma kn,0}}{P_{a_{\text{finish}-1},b_{\text{finish}-1}}}
\end{aligned} \tag{4.5}$$

This ratio $\mathcal{R}$ is also a product of many right-hand terms from (4.4), each smaller than 1, hence making the ratio exponentially small. All the terms in the ratio are non-zero since all states of the Markov Chain are reachable.

Unfortunately, Equations (4.3) do not hold. However, the cascading product (4.5) is still useful and meaningful if we replace the equalities of (4.3) by inequalities in the appropriate direction. We can build a path such that the inequalities are always in the direction of the next point in the path. Sadly, now we may have transitions where b increases or a decreases, that is, some of the terms in the cascading ratio are now larger than 1. Some algebra work overcomes that issue except when the start state – which we cannot choose anymore – is close to the edge of the grid. However, starting on the diagonal, we cannot be close to both bad sides, which solves this last issue. Note however that a situation where the starting point is not too close to both corners of the state space requires p to be moderately small. In our protocol where $p = f/k$ with f a constant, as well as in the other protocols that we are aware of, p is sufficiently small.

Nevertheless, one may want to design a protocol where p is a constant, that is, p is independent of the network size or view size. Such a value for p would not guarantee that the modified starting point is not poorly located in the state space. We believe our result on the exponential time before partitioning to hold true even with larger p's. The constraint on p came in when we had to consider the worst possible starting point for the modified path. This was seemingly due to a technical difficulty, not an intrinsic issue. For example, any reasonable starting point of the path, corresponding to a network relatively well balanced, lifts the requirement on p. Also, that requirement is dependent on the specifics of the equations capturing our protocol. The equations capturing a different protocol may well not require a bound on p.

The complete proof is detailed below.

4.3.1 Non-Negative-Flow Path

Non-Negative-Flow Path

Claim: There exists a path $\mathcal{P} = \{(a_0, b_0), (a_1, b_1), \ldots, (a_i, b_i), \ldots, (a_l, b_l)\}$ between any initial state (a_0, b_0) and any final state (a_l, b_l) such that, for each transition $(a_i, b_i) \rightarrow (a_{i+1}, b_{i+1})$ on the path, we have

$$P_{a_i,b_i} \Pr[(a_i, b_i) \rightarrow (a_{i+1}, b_{i+1})] \geq P_{a_{i+1},b_{i+1}} \Pr[(a_{i+1}, b_{i+1}) \rightarrow (a_i, b_i)]$$

This can be interpreted as having a non-negative (probability) flow between (a_i, b_i) and (a_{i+1}, b_{i+1}).

Proof: Consider the set S of states reachable from (a_0, b_0). Reachable means there exists a path from (a_0, b_0) satisfying our non-negative flow constraint. If S is not the whole set, there is a positive (probability) flow from the outside into S, since all transitions into S are positive. This situation violates the steady-state definition (no sinks, flows are at equilibrium).

Diagonal We want the ratios $P_{a+1,b}/P_{a,b}$ and $P_{a,b-1}/P_{a,b}$ in Equations (4.4) on page 49 to be less than one. These ratios are less than one for any point (a, b) placed above the diagonal line $a/\gamma = b/(1-\gamma)$. However, in order to simplify the algebra later on, we shall bound these ratios away from one. Define an upper parallel $\mathcal{D}$ to the diagonal $a/\gamma = b/(1-\gamma)$:

$$a = \frac{\gamma b}{1-\gamma} + \alpha \cdot \gamma k n \tag{4.6}$$

where α is a parameter smaller than 1. The optimum value of α may be $1/e$ but is of no interest to us. Any constant, say $1/2$, is good enough.

Returning to our path $\mathcal{P}$ discussion, we want it such that all reached states are above the diagonal $\mathcal{D}$ we just defined,

$$a \geq \frac{\gamma b}{1-\gamma} + \alpha \cdot \gamma kn \tag{4.7}$$

so that we can use the following inequality:

$$\frac{\gamma b}{(1-\gamma)a} \leq 1 - \alpha = 0.5 \tag{4.8}$$

Inequality (4.8) is derived from (4.7) using the fact that a is bounded by γkn.

Simple Path We choose a starting point (a_0, b_0) on $\mathcal{D}$ and consider a path $\mathcal{P}$ between (a_0, b_0) and $(\gamma kn, 0)$ on which the probability flow is always non-negative. We have just shown that such a path exists.

We simplify $\mathcal{P}$ by eliminating all loops in the path, if any. This does not change our flow property. Consider $\mathcal{P}'$, the portion of $\mathcal{P}$ from the last time $\mathcal{P}$ crosses the diagonal line $\mathcal{D}$ to the end. This ensures that $\mathcal{P}'s$ stays in the region above $\mathcal{D}$, that is, inequality (4.7) above holds for all the points on the path. Furthermore, this also ensures that $\mathcal{P}'$ does not spiral around the starting point. We shall detail now the main implication of this fact, which proves important for some later algebra to work out.

Lemma: Intersections Consider the intersection of $\mathcal{P}'$ with any horizontal line. The crossings alternate between up and down, from right to left on the intersecting line. In other words, if $\mathcal{P}'$ crosses the line going up at points A and B, then there is a point C *in-between* A and B where $\mathcal{P}'$ crosses the line going down.

Proof Assume $\mathcal{P}'$ goes to A first, then B, and consider the curve defined by the following three parts: $\mathcal{P}'$ from the start to A, the horizontal line segment A to

B, and $\mathcal{P}'$ from B to the end. This curve is a cut of the state space that $\mathcal{P}'$ needs to cross. The crossing part of $\mathcal{P}'$ is denoted by dots in Figure 4.4. By construction $\mathcal{P}'$ doesn't cross the cut in the first or third part (no loops). Hence $\mathcal{P}'$ crosses the segment A-B, going down. For any line higher than the starting point, the order

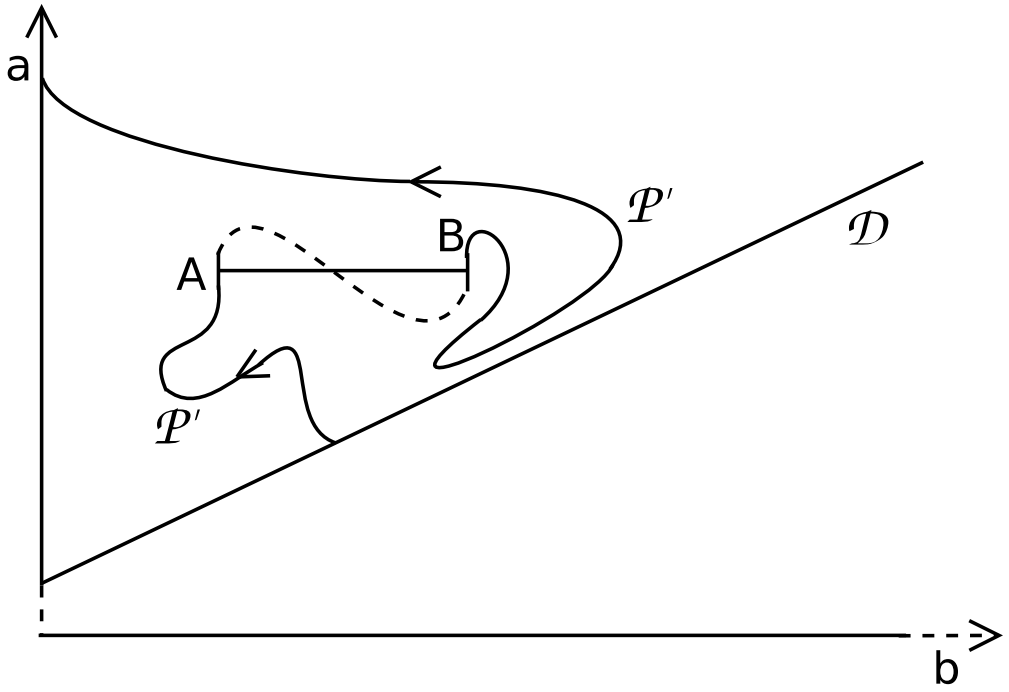


Figure 4.4: Intersection of $\mathcal{P}'$ with a Horizontal Line

of the crossings is obviously up, down up... up, since there is one more up than down. For lines below the starting point, we have up, down up... down (from left to right, that is, when b increases) and not the opposite. Add the diagonal from $(0, 0)$ to the starting point to $\mathcal{P}'$ into consideration for the proof of this fact.

Of course, the same can be proved for any vertical line: the intersection of $\mathcal{P}'$ with any vertical line is left right left right ... left from bottom to top. The first left is optional, depending on position of the line with respect to the starting point.

4.3.2 Ratio

We wish to study $\mathcal{R} = {}^{P_{\gamma kn,0}}/_{P_{a_{\text{start}},b_{\text{start}}}}$, ratio of the probability of the partitioned state to the start state:

$$\begin{aligned} \mathcal{R} &= \frac{P_{\gamma kn,0}}{P_{a'_0,b'_0}} \\ &= \prod_{(a_i,b_i)\in\mathcal{P}'} \frac{P_{a_{i+1},b_{i+1}}}{P_{a_i,b_i}} \\ &\leq \prod_{(a_i,b_i)\in\mathcal{P}'} \frac{\Pr[(a_i,b_i)\to(a_{i+1},b_{i+1})]}{\Pr[(a_{i+1},b_{i+1})\to(a_i,b_i)]} \end{aligned} \tag{4.9}$$

The inequality comes from our non-negative-flow path choice. Note the slight abuse of notation here: the last point in the path (the partitioned state) is not included in $\mathcal{P}'$ in the above product.

Since the path is constituted of steps on the grid, the only cases are a changes by one, or b changes by one; and we group the terms in the product following this distinction: all the vertical terms (b does not change during the transition) are in $\mathcal{R}_{\uparrow\downarrow}$, all the horizontal ones (a does not change during the transition) in $\mathcal{R}_{\leftrightarrows}$.

$$\begin{aligned} \mathcal{R}_{\uparrow\downarrow} &= \prod_{\substack{(a_i,b_i)\in\mathcal{P}' \\ b_i=b_{i+1}}} \frac{P_{a_{i+1},b_i}}{P_{a_i,b_i}} \leq \prod_{\substack{(a_i,b_i)\in\mathcal{P}' \\ b_i=b_{i+1}}} \frac{\Pr[(a_i,b_i)\to(a_{i+1},b_i)]}{\Pr[(a_{i+1},b_i)\to(a_i,b_i)]} \\ \mathcal{R}_{\leftrightarrows} &= \prod_{\substack{(a_i,b_i)\in\mathcal{P}' \\ a_i=a_{i+1}}} \frac{P_{a_i,b_{i+1}}}{P_{a_i,b_i}} \leq \prod_{\substack{(a_i,b_i)\in\mathcal{P}' \\ a_i=a_{i+1}}} \frac{\Pr[(a_i,b_i)\to(a_i,b_{i+1})]}{\Pr[(a_i,b_{i+1})\to(a_i,b_i)]} \end{aligned}$$

And we have

$$\mathcal{R} = \mathcal{R}_{\uparrow\downarrow}\cdot\mathcal{R}_{\leftrightarrows} \tag{4.10}$$

4.3.3 Up-Down Ratio

We first study $\mathcal{R}_{\uparrow\downarrow}$, using Equations (4.1) on page 47. Later, we shall regroup all the terms having the same a value together, canceling them with each other. We

define the up-down ratio along one transition $\mathcal{UP}(a,b)$: (Remember $c = \gamma kn - a$)

$$\mathcal{UP}(a,b) = \frac{\Pr[(a,b) \to (a+1,b)]}{\Pr[(a+1,b) \to (a,b)]} = \frac{c}{a+1} \cdot \frac{\frac{(1-p)b}{1-\gamma} + \frac{pa}{\gamma}}{\frac{(1-p)(c-1)}{\gamma} + \frac{pb}{\gamma}} \tag{4.11}$$

The ratio $\mathcal{UP}(a,b)$ is monotonic in b. Increasing almost always, decreasing only when c is small ($c \leq c_0 \approx n/k$).

Monotonicity

$$\begin{aligned}
\frac{d\mathcal{UP}(a,b)}{db} &= \frac{c}{a+1} \cdot \frac{\frac{(1-p)}{1-\gamma}\left(\frac{(1-p)(c-1)}{\gamma} + \frac{pb}{\gamma}\right) - \left(\frac{(1-p)b}{1-\gamma} + \frac{pa}{\gamma}\right)\frac{p}{\gamma}}{\left(\frac{(1-p)(c-1)}{\gamma} + \frac{pb}{\gamma}\right)^2} \\
&= \frac{c}{a+1} \cdot \frac{\frac{(1-p)^2(c-1)}{1-\gamma} - \frac{p^2 a}{\gamma}}{\gamma\left(\frac{(1-p)(c-1)}{\gamma} + \frac{pb}{\gamma}\right)^2}
\end{aligned}$$

The derivative ${}^{d\mathcal{UP}(a,b)}/_{db}$ is a constant with respect to b. It is positive for large values of c, that is $c > c_0$, and negative when $c < c_0$, where c_0 is such that:

$$\frac{(1-p)^2(c_0-1)}{1-\gamma} - \frac{p^2 a_0}{\gamma} = 0$$

Remembering that $a + c = \gamma kn$ and simplifying, we get:

$$\frac{(1-p)^2(c_0-1)}{1-\gamma} = p^2(kn - \frac{c_0}{\gamma})$$

Then

$$c_0 = \frac{p^2(1-\gamma)kn + (1-p)^2}{(1-p)^2 + p^2\,{}^{(1-\gamma)}/_{\gamma}} \approx \frac{p^2(1-\gamma)kn}{(1-p)^2} \leq \frac{f^2 n}{k} \tag{4.12}$$

The first derivative is proportional to

$$\frac{dc_0}{d\gamma} = Cst \times (p^4 kn\left(\frac{1-\gamma}{\gamma}\right)^2 + p^2(1-p)^2\left(\frac{1}{\gamma^2} - kn\right))$$

And the second derivative is proportional to

$$\frac{d^2 c_0}{d\gamma^2} = Cst \times -2(p^2 kn(1-\gamma) + \gamma(1-p)^2 kn) < 0$$

More algebra shows that when $k \gtrapprox \sqrt{n}$, then c_0 is decreasing as a function of γ. When $k \leq \sqrt{n}$, then c_0 is increasing up to $\gamma \approx p$, then decreasing. This does not have an effect on the maximum value of $c_{0,\max}$ which is approximatively p^2kn. The minimum value is $c_{0,\min} = 2$.

Dropping terms which are always dominated by others we get a usable expression for c_0:

$$2 \leq c_0 \approx \frac{(1-\gamma)knp^2}{1+p^2\frac{1}{\gamma}} \leq p^2kn = \mathcal{O}(\frac{n}{k}) \tag{4.13}$$

The $\mathcal{O}(\frac{n}{k})$ is justified only when the probability p of reinforcement is $p = f/k$ where f is a constant.

Combining the terms We combine all the $\mathcal{UP}(a,b)$ for a given a (or c), that is, we intersect the path $\mathcal{P}'$ with the horizontal line defined by the value a. (This of course assumes that the axes of the grid of the state diagram put a vertically, and b horizontally as we mentioned earlier.) We define the restriction of $\mathcal{R}_{\uparrow\downarrow}$ on that line:

$$\begin{aligned}
\mathcal{R}_{\uparrow\downarrow}(a) &= \prod_{\substack{b, \text{ such that} \\ \{(a,b)\to(a+1,b)\}\in\mathcal{P}'}} \frac{P_{a+1},b}{P_{a,b}} \prod_{\substack{b, \text{ such that} \\ \{(a+1,b)\to(a,b)\}\in\mathcal{P}'}} \frac{P_a,b}{P_{a+1,b}} \\
&\leq \prod_{\substack{b, \text{ such that} \\ \{(a,b)\to(a+1,b)\}\in\mathcal{P}'}} \mathcal{UP}(a,b) \prod_{\substack{b, \text{ such that} \\ \{(a+1,b)\to(a,b)\}\in\mathcal{P}'}} \frac{1}{\mathcal{UP}(a,b)}
\end{aligned}$$

and we have

$$\mathcal{R}_{\uparrow\downarrow} = \prod_{a=0}^{\gamma kn-1} \mathcal{R}_{\uparrow\downarrow}(a) \tag{4.14}$$

We saw earlier that the intersections between $\mathcal{P}'$ and a horizontal line alternate between up and down. An "up" means $\mathcal{UP}(a,b)$ in the bound of $\mathcal{R}_{\uparrow\downarrow}(a)$ and "down" means $1/\mathcal{UP}(a,b')$. Because $\mathcal{UP}(a,b)$ is monotonic in b, we combine the terms two by two to get a product of terms $\mathcal{UP}(a,b)/\mathcal{UP}(a,b')$ all less than 1. We are left with one

unpaired term for each $a \geq a_{\text{start}}$ and none for any $a < a_{\text{start}}$. The product of the unpaired terms, all smaller than one, yield our claimed exponential bound.

From now on, it is easier to work with c than a. Remember that $c = \gamma kn - a$, that is, c is the distance between a and the top of the grid. We compute a bound on $\mathcal{R}_{\uparrow\downarrow}(a)$. The bound essential to our result is Case 2 below yielding inequalities (4.16) and (4.17) on the following page. The rest can be trivially upper-bounded by a small polynomial in n. For illustration purposes we develop these bounds and set the probability of reinforcement to $p = 1/k$ to simplify the algebraic derivation. There are two cases, the good case when the starting point c_{start} of $\mathcal{P}$ is below c_0 ($c_{\text{start}} \geq c_0$), and the troubling case when the starting point c_{start} of $\mathcal{P}$ is above c_0 ($c_{\text{start}} \leq c_0$).

Good case When the starting point c_{start} of $\mathcal{P}$ is below c_0 ($c_{\text{start}} \geq c_0$).

1. For $c \geq c_{\text{start}}$, there is an even number of intersections, with an "up" for the smallest b value. We simply match every "up" with the next "down" and all these terms have ratios that are less than 1 since $\mathcal{UP}(a, b)$ is actually increasing in b.

2. For $c_0 \leq c < c_{\text{start}}$ there is an odd number of intersections. As above, we match every "up" with the next "down," yielding terms less than one since $\mathcal{UP}(a, b)$ is increasing here as well. We are left with a lonely "up" on the far right. This "up" is bounded by the $\mathcal{UP}(a, b)$ taken on the diagonal (as defined in (4.6) on page 51). We revisit the $\mathcal{UP}(a, b)$ expression as defined

in (4.11) on page 55:

$$\begin{aligned} \mathcal{UP}(a,b) &= \frac{c}{a+1} \cdot \frac{\frac{(k-1)b}{1-\gamma} + \frac{a}{\gamma}}{\frac{(k-1)(c-1)}{\gamma} + \frac{b}{\gamma}} \\ &\leq \frac{c}{a+1} \cdot \frac{\frac{(k-1)b}{1-\gamma} + \frac{a}{\gamma}}{\frac{(k-1)(c-1)}{\gamma}} \end{aligned} \tag{4.15}$$

Looking at the numerator (of the left part), if the $(k-1)b/1-\gamma$ is small compared to a/γ, say we have $(k-1)b/1-\gamma \leq k/3 \cdot a/\gamma$, then we get for (4.15):

$$\mathcal{UP}(a,b) \leq \frac{1}{3} \cdot \frac{k+2}{k-1} \cdot \frac{a}{a+1} \cdot \frac{c}{c-1} \tag{4.16}$$

Otherwise, we can neglect the left part and rewrite Equation (4.15) in

$$\begin{aligned} \mathcal{UP}(a,b) &\leq \frac{c}{a+1} \cdot \frac{\frac{(k-1)b}{1-\gamma}}{\frac{(k-1)(c-1)}{\gamma}} \\ &= \frac{c}{c-1} \cdot \frac{a}{a+1} \cdot \frac{\gamma b}{(1-\gamma)a} \\ &\leq (1-\alpha) \cdot \frac{c}{c-1} \cdot \frac{a}{a+1} \\ &\leq \frac{1}{2} \end{aligned} \tag{4.17}$$

the second last line coming from inequality (4.8) on page 52 and the last line from an appropriate choice of $\alpha = 1/2$.

3. For $c < c_0$ we match every "down" with the next "up," yielding terms less than one since $\mathcal{UP}(a,b)$ is decreasing. We are left with a lonely "up" on the far left, whose value is upper-bounded by

$$\mathcal{UP}(a, b=0) = \frac{1}{k-1} \cdot \frac{c}{c-1} \cdot \frac{a}{a+1} \tag{4.18}$$

Of course, this doesn't apply for $c = 1$, whose special treatment follows.

- If we reach the partitioned state going up, that is, stepping from $(a = \gamma kn - 1, b = 0)$ to $(a = \gamma kn, b = 0)$, then the $\mathcal{UP}$ ratio involves our hand-crafted

transition out of the partitioned state.

$$\mathcal{UP}(\gamma kn-1,0)=\frac{\Pr[(\gamma kn-1,0)\rightarrow(\gamma kn,0)]}{\Pr[(\gamma kn,0)\rightarrow(\gamma kn-1,0)]}=\frac{\frac{1}{k^2n}}{\frac{1}{2}}\leq 1 \tag{4.19}$$

- If we reach the partitioned state going left, that is, stepping from ($a=\gamma kn, b=1$) to ($a=\gamma kn, b=0$), then the $\mathcal{UP}$ ratio is upper-bounded by $\mathcal{UP}(a=\gamma kn, b=1)$. This is just another way of saying that we had to step up to $a=\gamma kn$ for some value of $b\geq 1$. We rework from (4.11) on page 55.

$$\mathcal{UP}(a,b)=\frac{1}{a+1}\cdot\frac{\frac{(k-1)b}{1-\gamma}+\frac{a}{\gamma}}{\frac{b}{\gamma}}$$

If the term a/γ dominates the numerator, we have

$$\mathcal{UP}(a,b)\leq 2\frac{1}{b}\cdot\frac{a}{a+1}\leq 2 \tag{4.20}$$

otherwise, when b is large, we have

$$\mathcal{UP}(a,b)\leq 2\frac{1}{\gamma kn}\cdot\frac{\gamma(k-1)}{1-\gamma}\leq\frac{2}{(1-\gamma)n}\leq\frac{2}{k} \tag{4.21}$$

since $(1-\gamma)\geq k/n$.

Conclusion:

$$\mathcal{R}_{\uparrow\downarrow}(a)\leq\begin{cases}2 \text{ for } c=1 & \text{see inequalities 4.19 to 4.21}\\ 1/2 \text{ for } 1<c<c_{\text{start}} & \text{see inequalities 4.16 to 4.18}\\ 1 \text{ for } c>c_{\text{start}}\end{cases}$$

And, combining all terms going up or down in ratio $\mathcal{R}$ of Equation (4.14) on page 56, we get

$$\mathcal{R}_{\uparrow\downarrow}\leq\left(\frac{1}{2}\right)^{c_{\text{start}}}$$

Troubling case: when the starting point c_{start} of $\mathcal{P}$ is above c_0 ($c_{\text{start}} \leq c_0$). We are still pairing two by two all the terms in Equation (4.9) on page 54 for a given c. There are issues for all $c_{\text{start}} < c < c_0$ due to $\mathcal{UP}$ decreasing.

1. For $c < c_{\text{start}}$, we can apply what we developed above in the third item of the good case for $c < c_0$.

2. For $c \geq c_0$, we can also apply what has been developed above in the first item of the good case ($c > c_{\text{start}}$)

3. For $c_{\text{start}} \leq c < c_0$, we have to use a different pairing: we pair the first term (which is an "up") with the last one (a "down"), then pair the second term with the third; the fourth with the fifth, etc. creating "down/up" pairs. Because $\mathcal{UP}$ is decreasing in b, this ensures that all the pairs except the first one yield ratios less than 1.

   The ratio $\mathcal{M}$ of the first to the last pairing is maximum when the "up" is completely on the left, and the "down" as far as possible on the right: the "up" is upper-bounded by $\mathcal{UP}(a, b = 0)$. The "down" is lower-bounded by $\mathcal{UP}(a, b^*)$ where b^* is such that (a, b^*) is on the upper-diagonal $\mathcal{D}$, verifying Equation (4.6) on page 51. Reusing Equation (4.11) on page 55, with suitable values, we have:

$$\begin{aligned}
\mathcal{UP}(a, b) &= \frac{c}{a+1} \cdot \frac{\frac{(k-1)b}{1-\gamma} + \frac{a}{\gamma}}{\frac{(k-1)(c-1)}{\gamma} + \frac{b}{\gamma}} \\
\mathcal{UP}(a, b = 0) &= \frac{1}{k-1} \cdot \frac{c}{a+1} \cdot \frac{a}{c-1} \\
\mathcal{UP}(a, b = (1-\gamma)kn) &= \frac{c}{a+1} \cdot \frac{\gamma(k-1)kn + a}{(k-1)(c-1) + (1-\gamma)kn}
\end{aligned}$$

Then we have:

$$\begin{aligned}
\mathcal{M} &\leq \frac{\mathcal{UP}(a, b=0)}{\mathcal{UP}(a, b=(1-\gamma)kn)} \\
&= \frac{a}{(k-1)(c-1)} \cdot \frac{(k-1)(c-1)+(1-\gamma)kn}{\gamma(k-1)kn+a} \\
&\leq \frac{\gamma kn}{k-1} \cdot \frac{(k-1)(c-1)+(1-\gamma)kn}{\gamma(k-1)kn} \qquad \text{(since } a \leq \gamma kn \text{ and } c \geq 2\text{)} \\
&\leq \frac{c+(1-\gamma)n}{k-1} \qquad (4.22)
\end{aligned}$$

Conclusion:

$$\mathcal{R}_{\uparrow\downarrow}(a) \leq \begin{cases} 2 \text{ for } c = 1 & \text{see inequalities 4.19 to 4.21} \\ 1/2 \text{ for } 1 < c < c_0 & \text{see inequalities 4.16 to 4.18} \\ (c+(1-\gamma)n)/(k-1) \text{ for } c_0 < c < c_{\text{start}} & \text{see inequality 4.22} \\ 1 \text{ for } c > c_0 & \end{cases}$$

Combining both cases:

$$\mathcal{R}_{\uparrow\downarrow} \leq \left(\frac{1}{2}\right)^{c_{\text{start}}} \left(\frac{c+(1-\gamma)n}{k-1}\right)^{\max(c_0-c_{\text{start}},0)}$$

4.3.4 Left-Right Ratio

By symmetry, we derive similar bounds as in the up-down case. It starts with

$$\frac{\Pr[(a,b) \to (a,b-1)]}{\Pr[(a,b-1) \to (a,b)]} = \frac{b}{d+1} \cdot \frac{\frac{(k-1)c}{\gamma} + \frac{d}{1-\gamma}}{\frac{(k-1)(b-1)}{1-\gamma} + \frac{c}{1-\gamma}}$$

Similar to (4.13) on page 56, we get

$$2 \leq b_0 \approx \frac{\gamma kn}{(k-1)^2 + \frac{1}{1-\gamma}} \leq \frac{n}{k} \qquad (4.23)$$

$$\mathcal{R}_{\leftrightarrows} \leq \left(\frac{1}{2}\right)^{b_{\text{start}}} \left(\frac{b+\gamma n}{k-1}\right)^{\max(b_0-b_{\text{start}},0)}$$

4.3.5 Combining Both Ratios

It is important to notice that we cannot have at the same time $b_{\text{start}} < b_0$ and $c_{\text{start}} < c_0$, since $(b_{\text{start}}, c_{\text{start}})$ is on $\mathcal{D}$ (see Equation (4.6) on page 51), that is

$$\frac{b_{\text{start}}}{(1-\gamma)} + \frac{c_{\text{start}}}{\gamma} = \frac{kn}{2}$$

Combining both up-down and left-right ratios, we get:

$$\mathcal{R} \leq \left(\frac{1}{2}\right)^{b_{\text{start}}+c_{\text{start}}} \left(1 + \frac{p}{1-p}\gamma kn\right)^{\max(b_0-b_{\text{start}},0)} \left(1 + \frac{p}{1-p}(1-\gamma)kn\right)^{\max(c_0-c_{\text{start}},0)}$$

At most one of the two right-most terms is raised to a non-zero power. When $\gamma \leq 1/2$, our bound is worst, that is, the above ratio is largest for $b_{\text{start}} = 0$, yielding $c_{\text{start}} = \gamma kn/2$. By symmetry for Equation (4.13), we have $b_0 = p^2\gamma kn$. The bound simplifies to:

$$\mathcal{R} \leq \left(\frac{(1 + \frac{p}{1-p}\gamma kn)^{p^2}}{\sqrt{2}}\right)^{\gamma kn}$$

As long as $p^2 \leq 1/\ln \gamma knp$ the upper term is negligible. This is the case since we set $p = f/k$ with f a constant.

$$\mathcal{R} \leq \left(\frac{1}{2}\right)^{\gamma kn/2}$$

Note that our bound on $\mathcal{R}$ is trivially an upper-bound on the limit probability of the partitioned state (see (4.9) on page 54): $P_{\gamma kn,0} = \mathcal{R}P_{a_{\text{start}},b_{\text{start}}} \leq \mathcal{R}$

4.3.6 Number of Partitions

The above bound has been computed for one partition where the sets are of size γn and $(1-\gamma)n$. The algebra below shows there are between $e^{-n\gamma \ln \gamma}$ and $e^{-2n\gamma \ln \gamma}$ of these. The exact number of these partitions is

$$\binom{n}{\gamma n} = \frac{n!}{(\gamma n)!((1-\gamma)n)!}$$

Using Stirling's approximation, we have

$$\binom{n}{\gamma n} \approx e^{-\gamma n \ln \gamma - (1-\gamma) n \ln(1-\gamma)}$$

The mathematically interesting part of this function is $\gamma \ln \gamma + (1-\gamma)\ln(1-\gamma)$. To know which term is negligible, we study the difference g, keeping in mind that there is an antisymmetry in $1/2$. Let $g(x) = x \ln x - (1-x)\ln(1-x)$. We have $g'(x) = 2 + \ln \frac{x}{1-x}$ and the following variation table:

x	0		$\frac{1}{e^2-1}$		$1/2$
$g'(x)$	$-\infty$	$-$	0	$+$	$+2$
$g(x)$	0	↘	$g(\frac{1}{e^2-1}) \leq 0$	↗	0

The $x \ln x$ term dominates the other one (for $x \leq 1/2$) and we conclude there are between $e^{-n\gamma \ln \gamma}$ and $e^{-2n\gamma \ln \gamma}$ partitions of size γn / $(1-\gamma)n$.

Conclusion

Using a union bound on all the partitions of size γn / $(1-\gamma)n$, we finally have that the probability of the system having a partition of such a size is less than:

$$P_{\text{partition of size } \gamma} \leq e^{-(k \ln \sqrt{2} + \ln \gamma)\gamma n}$$

For this bound to be meaningful, we need to have $k \geq \ln n$, since $-\ln \gamma$ can be almost as big as $\ln n$.

4.4 Proof, with Churn

We detail here the changes necessary to make the proof outlined in Section 4.3 apply when there is a churn rate of $\mu \neq 0$. In evaluating whether a set A of nodes

partitions away from the network, it makes sense to assume that these nodes don't die. In our proof, all the nodes joining and dying are B nodes, while the A nodes do not change.

4.4.1 Churn Model

The churn model is simple. At every round, μ randomly chosen nodes die and μ new nodes join the network. This choice was made in order to keep the number of nodes present in the system constant. One could consider a churn model wherein *expected value* μ nodes die. This would significantly add to the algebraic complexity of the derivation without much strengthening our results.

However simple, this churn model is also heavy-handed. We are assuming a constant number of nodes dying at *every single round*! This is not necessarily the case in real systems, especially if rounds happen quite often. See Chapter 6 for more details.

4.4.2 Handling Dead Edges

The definitions of a, b, c and d are unchanged. Let α be the number of dangling edges held in the views of A nodes and β number of dangling edges held by B nodes. We have $a + c + \alpha = \gamma kn$ and $b + d + \beta = (1 - \gamma)kn$.

We cannot analyze this 4-dimensional model: we do not know how to prevent the path $\mathcal{P}$ from spiraling around the start point, which breaks our previous analysis. Instead, we assume the fraction of dead edges to be constant: $\alpha = \lambda c$ and $\beta = \lambda b$. Edges pointing to A nodes cannot be dangling since the A nodes don't fail.

In steady state, the number of dangling edges removed and created at each

iteration of the protocol are equal. We have $\lambda = \frac{\mu}{pn}$.

4.4.3 Equations

There were $(1-p)kn$ pulls and pkn reinforcements per round. Now there are also $(b+c)\frac{\mu}{(1-\gamma)n}$ creations of dangling edges per round. Dangling edges are removed by the reinforcement process. In $\mathcal{B}$, the view of a failing node needs to be removed, replaced by the view of a joining node. It is easier to assume that the view of a failing node is taken over by a joining node. Not doing so would mean adding an extra term (insignificant in the end) to the following probabilities.

$$\Pr[(b,c) \to (b,c-1)] = \frac{\frac{c\mu}{(1-\gamma)n}}{kn + \frac{(b+c)\mu}{(1-\gamma)n}} + \frac{kn}{kn + \frac{(b+c)\mu}{(1-\gamma)n}} \left(p\frac{a}{kn}\frac{c}{\gamma kn} + (1-p)\frac{c}{kn}\frac{b+\beta}{(1-\gamma)kn} \right)$$

$$\Pr[(b,c) \to (b,c+1)] = \frac{kn}{kn + \frac{(b+c)\mu}{(1-\gamma)n}} \left(p\frac{b}{kn}\frac{a+\alpha}{\gamma kn} + (1-p)\frac{a}{kn}\frac{c}{\gamma kn} \right) \quad (4.24)$$

4.4.4 Up-Down Ratio

For $\mathcal{UP}(a,b)$, we now have: (Note that $a+c$ do not add up to γkn anymore.)

$$\mathcal{UP}(b,c) = \frac{p\frac{\lambda ac}{\gamma} + (1-p)\frac{bc}{(1-\gamma)(1+\lambda)} + (1-p)\frac{\lambda ckn}{1+\lambda} + \frac{ck\mu}{(1-\gamma)}}{(1-p)\frac{(a+1)(c-1)}{\gamma} + pb(kn - \frac{c-1}{\gamma})}$$

For our results to hold, $\mathcal{UP}(b,c)$ needs to be less than one, meaning that we are less likely to step toward the partitioned state than away. The main difference with Equation (4.11) on page 55 in Section 4.3 is the third and fourth terms of the numerator, corresponding to the creation of dangling edges. Fortunately they are small, otherwise the result would not hold.

The monotonicity of $\mathcal{UP}(a,.)$ changes direction at c_0 where $c_0 \approx \frac{(1-\gamma)p^2kn+\gamma k\mu}{(1-p)^2}$ By symmetry we get b_0:

$$b_0 \approx \frac{\gamma p^2 kn + (1-\gamma)k\mu}{(1-p)^2} \tag{4.25}$$

Again, our bound on $\mathcal{R}$ is worse (for $\gamma \leq 1/2$) when $c_{\text{start}} = \gamma kn/2$ and $b_{\text{start}} = 0$. The bound is:

$$\mathcal{R} \leq \frac{poly(n)^{b_0}}{2^{\gamma kn/2}} = \left(\frac{poly(n)^{b_0/\gamma kn}}{\sqrt{2}}\right)^{\gamma kn}$$

The ratio $b_0/\gamma kn$ needs to go to 0 for the bound to be small.[2] From (4.25), dropping irrelevant terms we have the claimed result when the following ration is small:

$$\frac{b_0}{\gamma kn} \approx \frac{\mu}{\gamma kn} \ll 1$$

4.4.5 Comment on Model:

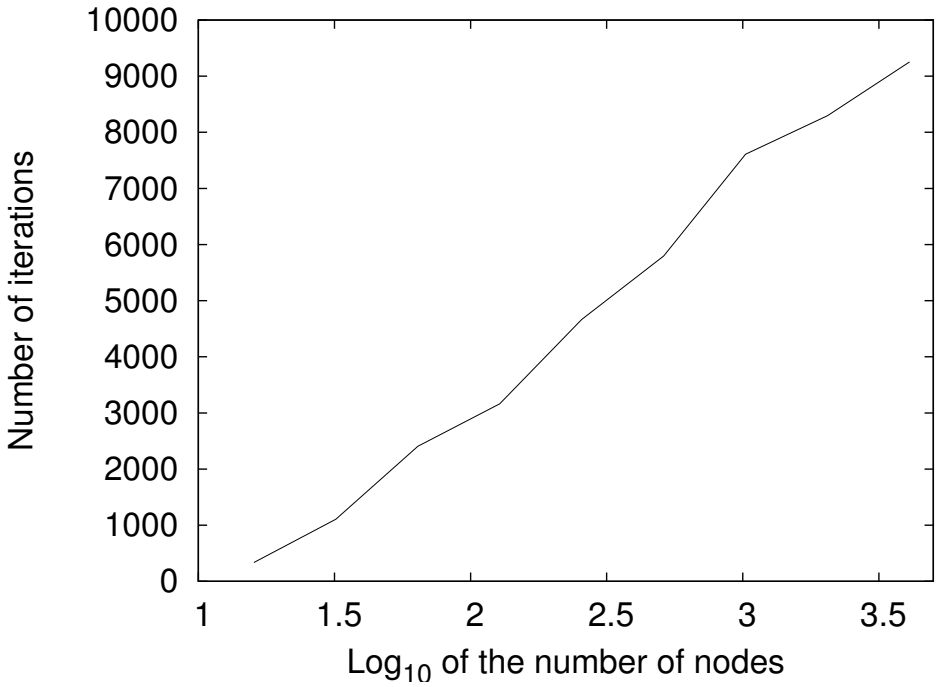
The case of nodes dying and joining in *both* sets of the partition is a straight forward modification. The third term of Equation (4.24) on the preceding page changes, and α and β are now constant. The algebra giving b_0 and c_0 is also simpler.

4.5 Simulations

We were interested in matching our theoretical results about partitioning and churn with simulation results. We ran simulations evaluating the number of iterations until partitioning. By partitioning, we include single nodes getting disconnected for having only dangling edges. Unsurprisingly, increasing view sizes or decreasing the churn increased the number of iterations before partitioning. More interestingly, so

[2]Being less than $C \ln n$ is sufficient where C is a constant.

did increasing the number of nodes *without* increasing the view size. For example, with no churn, the average number of iterations until partitioning with a view of 4 elements and a fanout of 2 was respectively 2×10^5, 3×10^6 and 7×10^6 for 16, 64 and 256 nodes. Due to the exponential nature of the phenomena, it is only possible to simulate for small view sizes and / or high churn rates. Also, increasing the fanout from 2 to 3 very significantly increased the time to partition. More details may be found in Appendix A.



Churn taken as $\mu = .32n$ where n is the number of nodes

Figure 4.5: Number of Iterations until Partitioning

In Figure 4.5, we verified the scaling properties of the protocol. With n the number of nodes in the system, we set the view size to $k = \log_2 n$ and the churn to $\mu = n/32$. Such a high churn is required to actually see partitioning in a reasonable time. Even though there were over a 120 runs for each point (except 40 for the last

one), the standard deviation was of the same order of magnitude as the average. The fact that the curve is increasing suggests that the system scales. It also suggests that the requirement $\mu \ll k$ of our theorem may not be necessary.

The size of the small partition decreases when the churn increases. It goes from an average of 5.8 when there is no churn down to 1 or 2 for high churn.

4.6 Other Algorithms

The same derivation can be made for the swap algorithm. The exact algebra slightly changes since probability Equations (4.1) on page 47 need to be modified to reflect the protocol. The modified equations are simpler. The derivation goes through and yields a similar result. The same holds for other protocols as well.

Chapter 5

In-degree Analysis

5.1 Introduction

For obvious practical reasons, we want the load to be spread relatively evenly among all the nodes of the network. We do not want a few nodes bearing the whole load by themselves; we do not want a few nodes being pulled by many. The load of a node is directly related to the in-degree of that node. In this chapter we study the in-degree distribution of the main protocol as well as the swap protocol.

The distribution of the node in-degree is governed by the way the pointers are created, copied and destroyed. Edges pointing to the various nodes are created at the same rate. The variability in the in-degree distribution comes from edge copying (some edges are copied more than others) and from the variability in life span. The latter is where almost all the in-degree variability lies. If all the edges pointing to node i live twice as long as the ones pointing to node j, node i's in-degree is going to be roughly twice as large as node j's in-degree.

The lifespan variability in our protocols is quite large due to the randomized nature of the choice of which edge to drop. Increasing reinforcement does decrease the lifespan variability. This is because increasing reinforcement increases the birth and death rate. Increasing reinforcement can be achieved by increasing the fanout in the main protocol or by using the swap protocol which has inherently much more reinforcement. Another alternative, not touched upon in this thesis, is to build a bias into the protocol so as to limit the lifespan variability. The simplest yet most efficient alternative is to keep track of the age of every edge and drop the oldest edges. Empirically this has been shown to achieve an excellent in-degree

distribution in [80]. Our techniques, while not directly applicable, do suggest why this is so.

5.1.1 Preliminaries

Each view consists of k distinct nodes. Consider the concatenation $\mathcal{V}$ of all the views. This list contains kn nodes. Ideally, each node would appear exactly k times. This is not the case though, some nodes appear more times, some less. The number of times node u appears in $\mathcal{V}$, which is the same as the number of nodes having node u in their view, is the best indicator of how many times node u will be pulled in one iteration of the protocol.

The protocol updates the list $\mathcal{V}$ at each iteration. The list at the next iteration is obtained by a complex sampling from the previous version of the list, reflecting the details of the protocol. We are unable to analyze the process updating $\mathcal{V}$ for two reasons:

1. the different elements of a view are not independent variables: indeed, they are constrained to be values from the list the node created during the update process.

2. some views are more likely to be requested than others, implying that the nodes present in these views are more likely than the nodes from other views to be included in a view at the next iteration.

Therefore, we introduce a simplified model which does not exhibit the above two difficulties, but still captures the essence of the protocol.

5.1.2 Simplified Model

We have an array of length kn representing the concatenation of all views. Each array element is (independently) updated according to the following:

- with probability $1-p$ replace the array element with the value of an element chosen uniformly at random from the array; this corresponds to the mixing of the views.

- with probability p replace the array element with a node uniformly at random, that is, randomly pick a number between 1 and n if there are n nodes; this corresponds to the reinforcement,

Synchronous Model: all elements are updated at the same time;

Asynchronous Model: in each round, a single element chosen uniformly at random is updated. Roughly, kn iterations of this model corresponds to one iteration of the the synchronous model.

From a protocol point of view, each node updates each element of its view independently. For almost every element in the view, the node updates it by picking a view from some node, (any node, not necessarily a node from its own view), and randomly choosing an element from that view. With some small probability p, it doesn't do so, but instead reinforces some node by picking at random a node from the list of all nodes in the system.

From a graph point of view: the pulling part is about moving (and duplicating) the tail of any edge without restriction, and the reinforcement plainly creates a new edge.

5.1.3 Model Justification

We made two assumptions which allowed us to derive our simplified model:

1. the view structure is irrelevant here;

2. the probability of each element of a view has the same probability of being selected.

The view structure is relevant with respect to the risk of partitioning. It is, however, irrelevant when studying the number of copies of each node in the system, since we do not care about which nodes hold these copies, only how many of them there are. This is why we completely forgo the view structure.

With respect to Point 2, the different copies of a given node have different probabilities of being selected for the next iteration, depending on which view they are present in. However, there is no correlation between a view and its content, and these probabilities are *independent* of the number of copies of each node.

We are replacing by a uniform sampling a sampling where the probabilities for each element are different, but independent of the element, and independently changing over time. This is a legitimate simplification. There is no divergence happening; instead, an over-representation of a node (and the increase in probability for the nodes of its view) is smoothed out rapidly:

Suppose we have $m \gg k$ copies of some node. The view of this node will be, in expectation, pulled $m * f/k$ times. At the next iteration, there will be $m/k \ll m$ extra copies of each of the k elements of that view. The view held by of each of these k elements will be pulled, in expectation, $m * (f/k)^2 << m * f/k$ times. At the following iteration, there will be $m/k^2 \ll m$ extra copies of each of the k^2 elements of these views, etc. In other words, a probability imbalance is evened out

in logarithmic many steps, and uniformity over all copies of all nodes is the correct assumption to make in our simplified model.

5.2 Without Reinforcement

Reinforcement is crucial to the functioning of the protocol. Indeed, we shall show that without reinforcement, many nodes quickly disappear from the system. They do not partition out, but they do not get pulled anymore. In other words, all the edges point to a small number of nodes – the few elements left in the union of the views – which bear the whole load by themselves. This is a star network.

No reinforcement means setting $p = 0$ in the previous model. From a graph point of view: we are only moving the tail of any edge to anywhere, possibly duplicating it.

5.2.1 Synchronous Model

Remember that we have an array $\mathcal{V}$ of length kn representing the concatenation of the views of all the nodes. The next-iteration array is obtained by selecting uniformly at random with replacement from the previous-iteration array.

We are interested in the number of distinct numbers in the array. Let $m(t)$ denote this number. We start with $m(0) = n$. Let $\#(i)$ denote the number of copies of the number i in our array at the beginning of iteration t, and $S_t = \{j \in [1..n] \mid \#(j) \neq 0\}$ denote the set of numbers present in the array at the start of iteration t. We have

$$\sum_{i \in S_t} \#(i) = kn$$

We want to look at the expected loss E_t in distinct elements during iteration t.

Let X_i be the binary random variable taking value 1 if number i is absent from the array $\mathcal{V}$ at the end of the round, and 0 otherwise. We have

$$E[X_i] = \left(1 - \frac{\#(i)}{kn}\right)^{kn}$$

and

$$\begin{aligned} E_t &= \sum_{i \in S_t} E[X_i] \\ &= \sum_{i \in S_t} \left(1 - \frac{\#(i)}{kn}\right)^{kn} \\ &\geq m(t) \left(1 - \frac{\sum_{i \in S_t} \#(i)}{m(t) * kn}\right)^{kn} \\ &\geq m(t) \left(1 - \frac{1}{m(t)}\right)^{kn} \\ &\geq m(t) e^{-\frac{kn}{m(t)}} \end{aligned}$$

The first inequality comes from the convexity of $(1 - x/kn)^{kn}$.

Lemma ([46]) *The time $T(X_0)$ needed for a non-increasing real-valued Markov chain $X_0, X_1, X_2 \ldots$ to drop to 1 is bounded by*

$$T(X_0) \leq \int_1^{X_0} \frac{1}{\mu_z} dz$$

when $\mu_z = E[X_t - X_{t+1} | X_t = z]$ is a non-decreasing function of z.

The proof can also be found in [66] p. 15 and a nice interpretation in [2] Section 4. Average speed is the ratio between traveled distance and duration. An upper bound on the speed yields a lower bound on travel duration.

We use a variation of cited lemma to bounding to time it takes to drop to l (instead of 1) distinct numbers in the array. Quantity $m(t)$ corresponds to X_t and

μ_z to E_t with $\mu_z \geq ze^{-\frac{kn}{z}}$. We have

$$\begin{aligned} E[T] &\leq \int_l^n \frac{e^{\frac{kn}{z}}}{z} dz \\ &\leq \int_l^n \frac{n}{z} \frac{e^{\frac{kn}{z}}}{z^2} dz \\ &\leq n \int_l^n \frac{e^{\frac{kn}{z}}}{z^2} dz \\ &\leq \frac{1}{k} e^{\frac{kn}{l}} \end{aligned}$$

This bound can be substantially improved because a lot is lost in the convexity argument: it is equivalent to assuming that no node behaves worse than the mean, which clearly is not the case.

However, we cannot prove anything stronger, and this illustrates how difficult it is to analyze the synchronous model.

5.2.2 Asynchronous Model

We turn our attention to the asynchronous model where a single element is updated at each round. Here we consider a partition of the nodes into two sets A and B, and evaluate the time it takes until no node from one set is left in any of the views. This means that all the nodes in A (or B) have in-degree 0.

We use the array $\mathcal{V}$ of length $m = kn$ as in the previous section, with the same update rule, except only one element, chosen uniformly at random, is updated per iteration. In the array, values are either 0 or 1 depending on which set of the partition a node is in.

Let $\Pr[i \to j]$ represent the probability of there being j zeros at the end of the iteration given that there are i zeros at the beginning on the iteration. Let $E[i]$ be the expected number of iterations before all zeros (or all ones) have disappeared when starting with i zeros. We have $E[0] = 0 = E[m]$.

The probability equations are as follow, for $0 < i < m$:

$$\begin{aligned} \Pr[i \to i+1] &= \frac{i(m-i)}{m^2} \\ \Pr[i \to i] &= (\frac{i}{m})^2 + (\frac{m-i}{m})^2 \\ \Pr[i \to i-1] &= \frac{i(m-i)}{m^2} \end{aligned}$$

And the equations governing the expectation are then $(0 < i < m)$:

$$E[i] = 1 + \frac{i(m-i)}{m^2}E[i+1] + \left((\frac{i}{m})^2 + (\frac{m-i}{m})^2\right)E[i] + \frac{i(m-i)}{m^2}E[i-1]$$

which can be written as

$$E[i+1] = 2E[i] - E[i-1] - \frac{m^2}{i(m-i)} \tag{5.1}$$

The solution to Equation 5.1 is of the form:

$$E[i] = iE[1] - m^2 q_i \tag{5.2}$$

Substituting $iE[1] - m^2 q_i$ for $E[i]$ in equation 5.1 yields

$$\begin{aligned} q_{i+1} &= 2q_i - q_{i-1} + \frac{1}{i(m-i)} \\ \text{with} \quad q_0 &= q_1 = 0 \end{aligned}$$

Expanding the formula for q_i yields:

$$q_i = \frac{i-1}{m-1} + \frac{i-2}{2(m-2)} + \frac{i-3}{3(m-3)} + \ldots \frac{i-j}{j(m-j)} + \ldots + \frac{1}{(i-1)(m-i+1)}$$

which, for $i = m$, simplifies to the harmonic series:

$$q_m = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{m} \approx \ln m$$

We now find the value of $E[1]$ since we have $E[m] = mE[1] - m^2 q_m = 0$, and

$$\begin{aligned} E[1] &= m \ln m \\ E[i] &= im \ln m - m^2 q_i \\ E[i] &\leq im \ln m \end{aligned}$$

Conclusion: in expectation, in the model where a single array element is updated at each iteration all the nodes from one side of the partition disappear from all the views in $\frac{1}{2}(kn)^2 \ln kn$ iterations. However, this is the time it takes for *our chosen set* of nodes to disappear. Other nodes might have already disappeared as well. Obviously, the time it takes for half the nodes (any nodes) to disappear is less.

Simulations of the protocol indicate a time sub-linear in m. Remember that m iterations of this asynchronous model correspond to one iteration in the synchronous model and the real protocol. So this bound is not tight by a factor of m. Indeed, from our simulations, it looks like the number of iterations before more than 90 % of the nodes have disappeared from the views[1] is logarithmic in n.

5.2.3 Conclusion

Reinforcement is a required component of any gossip-based membership management protocol. We believe the protocol of [74] to be accurately captured by our model with no reinforcement. As such, after the initial phase where the views are filled, the network will rapidly converge to a star, with a core whose size is the view size.

In fact, any membership protocol which re-samples randomly from the views without adding the names of the nodes currently in the system in some way or another is doomed to collapse. Consider $\mathcal{V}$, the concatenation of all the views. Iterating the protocol once corresponds to creating a new $\mathcal{V}$ by some kind of sampling with replacement from the old $\mathcal{V}$. Some nodes might disappear from $\mathcal{V}$ at each iteration. Once a node has disappeared, it cannot reappear without an ex-

[1] We only simulated with views of logarithmic size; fanout didn't have any effect.

ternal mechanism like reinforcement. The diversity of the content of $\mathcal{V}$ decreases over time, and in fact rather rapidly, leading to a star-like network. Note that it is theoretically possible to evade the issue by creating a protocol which would correspond to a permutation on $\mathcal{V}$, but this is rather tricky to implement, and doesn't necessarily behave nicely in the presence of nodes leaving or joining the network. Otherwise, one needs to actively add the names of the nodes currently in the network to $\mathcal{V}$, a process we call reinforcement. LwPBCast [22] has some reinforcement, even though not specifically mentioned in the article: each process adds itself to the "subs" field when sending a message. The same holds for Newscast [42] and Cyclon [80] as well: nodes add their own address to their view that they then disseminate.

Note the following interesting behavior in the context of news propagation [42]. Assume all the "News Events" are created by some subset $\mathcal{S}$ of the nodes. Let only the nodes creating "News Events" reinforce: nodes add their names to their views when creating a "News Events" instead of every T seconds. Then the network will converge to having a core ($\mathcal{S}$, the nodes creating messages) and a fringe (the other nodes). Were these nodes to change, the star would adapt and recenter itself on the nodes currently emitting messages.

5.3 Asynchronous Model with Reinforcement

To simplify the analysis, we consider a partition A-B of the nodes. A 0 denotes a node in set A, and a 1 denotes a node in set B. The only modification here compared to the model described in 5.1.2 is to have the array elements be either 0 or 1. The probability q equals the proportion of A nodes, and p is the probability of reinforcement. The number of zeros in the array (denoted by i) represents the

sum of the A-nodes' in-degree. No zeros means all the edges point to B nodes. All zeros means all the edges point to A nodes.

5.3.1 Equations

Individual Probabilities

Reinforcement Select uniformly at random an element from the array. With probability p, do reinforcement by setting the element's value to 0 with probability q and 1 otherwise. Thus:

$$\begin{aligned} \Pr[i \to i+1] &= q\frac{kn-i}{kn} \\ \Pr[i \to i] &= q\frac{i}{kn} + (1-q)\frac{kn-i}{kn} \\ \Pr[i \to i-1] &= (1-q)\frac{i}{kn} \end{aligned}$$

Mixing With probability $1-p$, an element copies its value from somebody else:

$$\begin{aligned} \Pr[i \to i+1] &= \frac{i(kn-i)}{(kn)^2} \\ \Pr[i \to i] &= (\frac{i}{kn})^2 + (\frac{kn-i}{kn})^2 \\ \Pr[i \to i-1] &= \frac{i(kn-i)}{(kn)^2} \end{aligned}$$

Overall Probabilities Putting things together:

$$\begin{aligned} \Pr[i \to i+1] &= pq\frac{kn-i}{kn} + (1-p)\frac{i(kn-i)}{(kn)^2} \\ \Pr[i \to i] &= p(\frac{i}{kn} + (1-q)\frac{kn-i}{kn}) + (1-p)((\frac{i}{kn})^2 + (\frac{kn-i}{kn})^2) \\ \Pr[i \to i-1] &= p(1-q)\frac{i}{kn} + (1-p)\frac{i(kn-i)}{(kn)^2} \end{aligned}$$

Limit Distribution

The limit distribution is defined by the following set of equations, for all $0 \leq i < kn$, where P_i is the steady state probability of having exactly i zeros in the array.

$$\sum_{l \leq i, m \geq i+1} P_l \Pr[l \to m] = \sum_{l \leq i, m \geq i+1} P_l \Pr[m \to l]$$

This denotes the flow equilibrium across a cut between entries i or less, and those above. Note that due to symmetry reasons, $P_i = P_{kn-i}$ However, $\Pr[l \to m] = 0$ whenever $|m - l| > 1$. Those equations simplify to

$$P_i \Pr[i \to i+1] = P_{i+1} \Pr[i+1 \to i] \tag{5.3}$$

As a side note, the same derivation can be made in the synchronous model. However, it would yield a formula giving P_i as a function of all the other P_j with $j = 1..n$ and not P_i as a function of just P_{i+1}. Having P_i as a function of all the other P_j makes it impossible to solve for P_i.

Equation 5.3 can be developed into the following:

$$\frac{P_{i+1}}{P_i} = \frac{kn - i}{i+1} \times \frac{pq + \frac{i}{kn}(1-p)}{1 - pq - \frac{i+1}{kn}(1-p)} \tag{5.4}$$

which can be rewritten so that

$$\frac{P_i}{P_0} = \prod_{j=1}^{i} \frac{kn + 1 - j}{j} \prod_{j=1}^{i} \frac{pq + \frac{i-1}{kn}(1-p)}{1 - pq - \frac{i-1}{kn}(1-p)} \tag{5.5}$$

$$= \binom{kn}{i} \frac{pq}{1 - pq - \frac{i}{kn}(1-p)} \prod_{j=1}^{i-1} \frac{pq + \frac{j}{kn}(1-p)}{1 - pq - \frac{j}{kn}(1-p)} \tag{5.6}$$

5.3.2 Analysis

First, let us check that the P_i's are actually increasing to a maximum value, then decreasing. We start from equation 5.4. Instead of showing that the ratio is

larger than 1, then smaller, we show that the numerator minus the denominator is positive, then negative.

$$\frac{P_{i+1}}{P_i} = \frac{kn-i}{i+1} \times \frac{pq + \frac{i}{kn}(1-p)}{1 - pq - \frac{i+1}{kn}(1-p)}$$

$$\begin{aligned} D(i) &= (kn-i)\left(pq + \frac{i}{kn}(1-p)\right) - (i+1)\left(1 - pq - \frac{i+1}{kn}(1-p)\right) \\ &= (kn - i + i + 1)pq + (1-p)\left((1 - \frac{i}{kn}i) + \frac{(i+1)^2}{kn}\right) - i - 1 \\ &= (kn+1)pq - 1 - pi + \frac{1}{kn}(1-p)(2i+1) \\ \frac{dD}{di} &= -p + 2\frac{1-p}{kn} = \frac{1}{kn}\left(2 - p(kn+2)\right) \end{aligned}$$

Clearly, D is monotonic. And it is increasing if and only if

$$p \geq p_0 = \frac{2}{kn+2}$$

In other words, in the case when $p > p_0$, P_{i+1}/P_i is increasing up to some $i_{\max}$, then decreasing from then on. $i_{\max}$ is the most likely number of zeros in the array.

The quantity $i_{\max}$ is obtained when $P_{i+1} = P_i$, that is, when $D = 0$. This gives us

$$i_{\max} = \frac{p[(kn+1)q - \frac{1}{kn}] - \frac{kn-1}{kn}}{p(1 + \frac{2}{kn}) - \frac{2}{kn}} \tag{5.7}$$

5.3.3 Real Values

Let us apply some reasonable values for the probabilities p and q, in order to get some approximations for $i_{\max}$. In the protocol, we have $p = \frac{1}{k}$. Note that this value is much larger than the $\frac{2}{kn+2}$ we need. However, simple tweaks could increase p to almost $\frac{f}{k}$, where f is the number of views requested by a node during its update.

So, with $1 \leq \eta < f$, let

$$p = \frac{\eta}{k}$$

Let λ denote the number of nodes in the partition we consider. Then we have, with $\lambda \in [1..n-1]$

$$q = \frac{\lambda}{n}$$

We expect that the most likely number of zeros in the array ($i_{\max}$) is approximatively the product of the size of the partition (λ) by the number of times each node is represented ($k = |\text{View}|$); that is: $i_{\max} \approx \lambda k$. This is indeed close to reality, in all but one case:

$$i_{\max} \approx \frac{1}{n} \qquad \text{when } \lambda = \eta = 1 \tag{5.8}$$

$$i_{\max} \approx k(\lambda - \frac{1}{\eta}) \qquad \text{otherwise} \tag{5.9}$$

Estimating the Number of Nodes with High or Low In-degree

The limit probability distribution looks like the curve of Figure 5.1 on the following page. The partition we considered is very likely going to be close to the pointy part of the curve. However, there isn't just one, but $\binom{n}{\lambda}$ partitions with one side of size λ. These $\binom{n}{\lambda}$ partitions will all be distributed according to the same curve. This leads to the following qualitative argument: when $P_{i_{\max}} >> \binom{n}{\lambda} P_0$, it is likely that none of these $\binom{n}{\lambda}$ partitions is such that all the edges point to one side of the partition. However, when $P_{i_{\max}} \approx \binom{n}{\lambda} P_0$, it is likely that at least one of the partitions is such that all the edges point to one side of the partition. This yields an estimate on the number of nodes with no edges pointing to them, and an estimate on the number of nodes with many nodes pointing to them.

Consider the case of nodes with no in-degree. Setting $\lambda = 1$ in the model corresponds to a single node A in one set of the partition, and the other $n - 1$

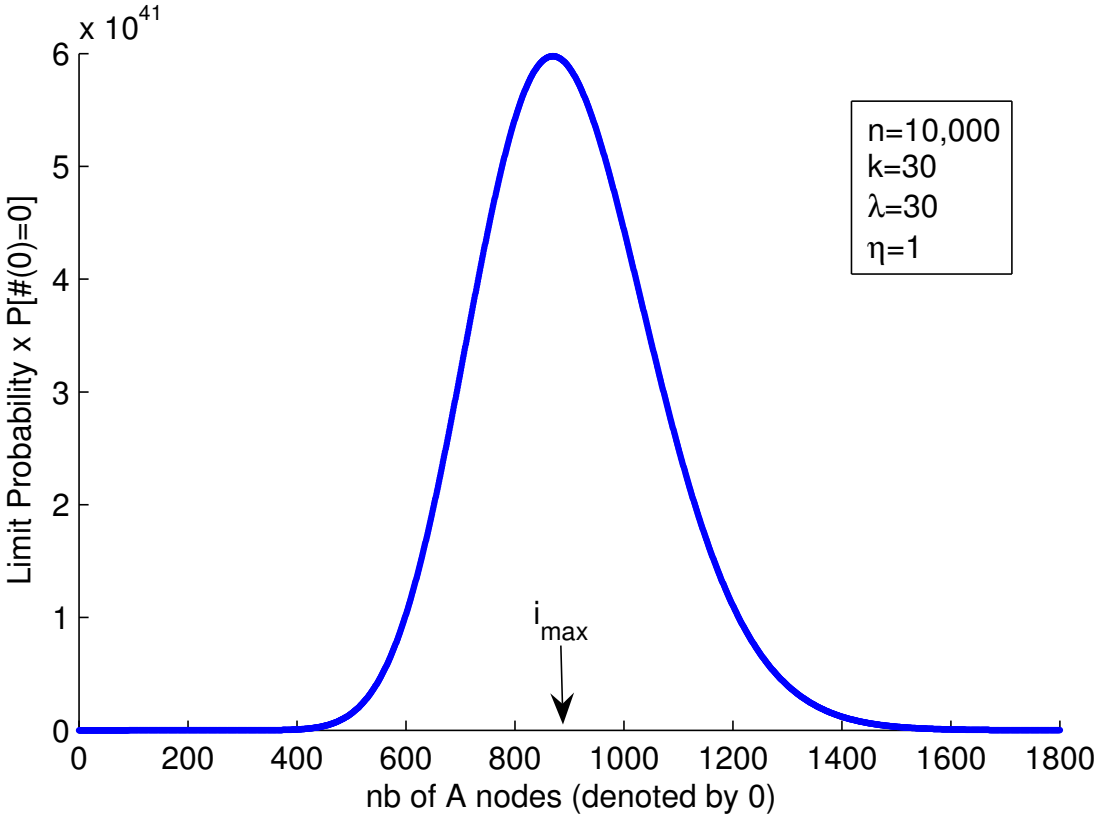


Figure 5.1: In-degree Probability Distribution

in the other set of the partition. The probability P_0 is the probability that the in-degree of node A is 0. The expected number of nodes with no in-degree is nP_0. Table 5.1 on the following page shows some of the numerical values that were obtained. As mentioned earlier, it is difficult to know precisely what value of η corresponds to each value of the fanout f, this is why two values of η are presented for each f in the table. While the predictions constantly underestimate the number of nodes with no in-degree, the order of magnitude is always correct.

Table 5.1: Predictions of the Number of Nodes with No In-degree

Number of nodes with no in-degree					
Simulations		Predictions			
fanout = 1	9,300	$\eta = 1$	6,767	$\eta = 0.9$	8,7000
fanout = 3	740	$\eta = 3$	238	$\eta = 2.5$	500
fanout = 5	220	$\eta = 5$	22	$\eta = 4$	68

Number of nodes $n = 2^{17} = 131,072$; View size $k = 17$

Simulations

Simulations for large numbers of nodes have shown that the performance of the protocol is quite good. For $2^{17} \approx 100,000$ nodes, view sizes of 17, a fanout of 3 and a loosely synchronized system, the maximum in-degree was always below 4.5 times that of a random graph and the standard deviation was not more than 3.2 times larger than that of a random graph. These values would improve with increased fanout, but even a fanout of 2 gives satisfactory performance.

5.4 Swap Protocol

The in-degree distribution is significantly simpler to analyze in the swap protocol. We shall consider two models. In the first one, a randomly chosen node initiates the update procedure. Then, another randomly chosen node (potentially the same) initiates the update procedure, etc. This is the asynchronous model. In the synchronous model, a random ordering of the nodes is chosen, then, each node successively in that order initiates the update procedure. In other words, every node gets to initiate the procedure once per cycle of n updates.

Similarly to the other protocol, the asynchronous model is easier to analyze. It is standard practice to use an asynchronous model to analyze a synchronous protocol since the analysis would otherwise be intractable. Simulations typically support this simplification and do not exhibit any significant difference. Here however, the difference between the in-degree distribution of the synchronous swap protocol and of the asynchronous swap protocol is quite significant: the standard deviation is twice as large in the latter case, though still much smaller than that in the main protocol.

5.4.1 Balls and Bins Formulation

The swap protocol can be reformulated in the following way in terms of balls and bins. Each node corresponds to a bin, each edge to a ball. Each ball in bin i denotes an edge pointing the node i. The number of balls in bin i is node i's in-degree. Initially the balls are distributed uniformly at random in the bins.

Swap Protocol, Asynchronous

Forever do:

1. Select a bin uniformly at random among all bins.
2. Select a ball uniformly at random among all balls.
3. Put the ball in the bin.

End forever.

For reference, the swap protocol is detailed on Figure 2.3 on page 23. Only two nodes out of the three involved in the update see their in-degree change. Remember, node A asked node B for an edge (to some node C). Node A replaces its edge to B by an edge to C, node B replaces its edge to C by an edge to A.

This is the same as having node A take over B's edge to node C and flipping the direction of the edge from A to B. The in-degree of node C is unchanged by the update operation. Node A, whose in-degree increases by one, is the node chosen for update. It is chosen uniformly at random in the asynchronous model. Node B, whose in-degree decreases, is chosen proportionally to its in-degree, assuming the edge distribution to be random.

Swap Protocol, Synchronous

Forever do:

1. Choose a random ordering of the n bins.
2. For each bin in the chosen order do:
    1. Select a ball uniformly at random among all balls.
    2. Move the ball to the selected bin/

    End for.

End forever.

5.4.2 In-degree Distribution in the Asynchronous Case

Consider a single node. The model is the following: with probability $1/n$, the node is chosen to initiate the update and sees its in-degree increase by one. With probability proportional to its in-degree, $p = {}^{in-degree}/_{kn}$, the node acts as node B in the swap protocol and its in-degree decreases by 1. See Figure 5.2 on the following page for a graphical representation of the Markov Chain.

This Markov Chain is easily analyzed. Let P_i the limit probability distribution of state i (corresponding to i balls in the bin). Writing the steady state equilibrium

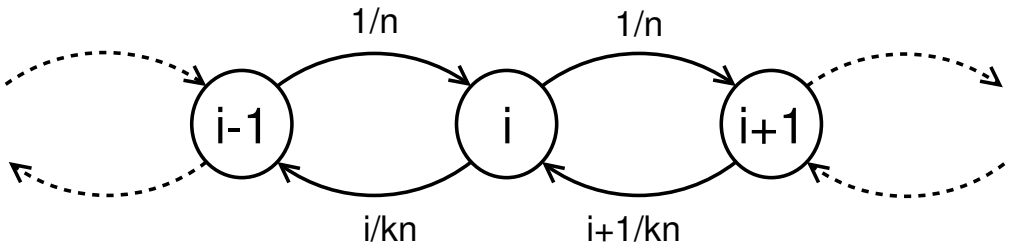


Figure 5.2: Markov Chain for the Asynchronous Model

equation across nodes $i-1$ and i we have:

$$\frac{1}{n}P_{i-1} = \frac{i}{kn}P_i$$

The factor $1/n$ simplifies out. Cascading all the way to $i = 1$, we get

$$P_i = \frac{i!}{k^i}P_0 \tag{5.10}$$

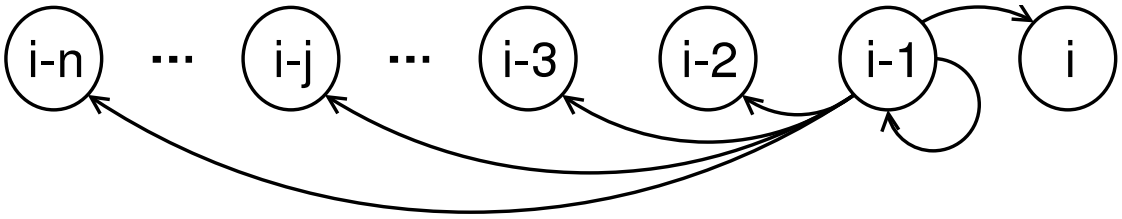
This is the Poisson distribution, whose standard deviation is $\sigma = k$.

The model neglected the correlation with the other nodes in the system: when the node's in-degree increase, some node's in-degree decreases. This is a negative correlation. Reference [19] should allow us to conclude that this model gives a rigorous upper bound to the asynchronous protocol.

5.4.3 In-degree Distribution in the Synchronous case

Model and Probabilities

We use a similar model, but the Markov Chain is more complex. Consider what happens to one node during one cycle. It initiates an update once, increasing its in-degree by one. It then has n chances to be selected for a swap, leading to n opportunities of decreasing its in-degree by one. The probability of being selected



Only the transitions out of state i are shown.

Figure 5.3: Markov Chain for the Synchronous Model

for a swap is i/kn when the current in-degree is i. The resulting state transitions are shown on Figure 5.3. The probability of these transitions are the following:

$$\begin{aligned}
\Pr[i-1 \to i] &= \left(1-\frac{i}{kn}\right)^n \\
\Pr[i-1 \to i-1] &= \sum_{t=0}^{t=n} \left(1-\frac{i}{kn}\right)^t \frac{i}{kn} \cdot \left(1-\frac{i-1}{kn}\right)^{n-t-1} \\
&\cdots \\
\Pr[i-1 \to i-j] &= \sum_{\substack{t_1,t_2,\ldots,t_j \\ \sum_{l=1}^{j} t_l + j \le n}} \left(1-\frac{i}{kn}\right)^{t_1} \frac{i}{kn} \cdot \left(1-\frac{i-1}{kn}\right)^{t_2} \frac{i-1}{kn} \cdots \\
&\qquad \cdot \left(1-\frac{i-j+1}{kn}\right)^{t_j} \frac{i-j+1}{kn} \cdot \left(1-\frac{i-j}{kn}\right)^{n-j-\sum t_l} \\
\Pr[i-1 \to i-n] &= \frac{i}{kn}\frac{i-1}{kn}\cdots\frac{i-j}{kn}\cdots\frac{i-n+1}{kn} \qquad (5.11)
\end{aligned}$$

Consider $\Pr[i-1 \to i-j]$. The node is selected j times for a swap. Let t_l denote the number of iterations between the $l-1$th and the lth time the node is selected for a swap. The node in-degree changes from $i-l+1$ to $i-l$ at iteration $\sum_{m \le l}(t_m+1) = \sum_{m \le l} t_m + l$. The probability of the node's in-degree not changing during these t_l iterations is $\left(1-\frac{i-l+1}{kn}\right)^{t_l}$: none of the edges pointing to the node are selected for a swap and the selection process is repeated t_l times. Then the node is selected, which happens with probability $(i-l+1)/kn$ since the current in-degree is $i-l+1$. Once the node has been selected j times, it is not selected anymore.

There are $n-\sum t_l-j$ iterations left and the probability is $\left(1-\frac{i-l+1}{kn}\right)^{n-j-\sum t_l}$. The probability of the node's in-degree decreasing j times, at iterations $\sum_{l\leq m} t_m + l$, is:

$$P = \left(1-\frac{i}{kn}\right)^{t_1}\frac{i}{kn}\cdot\left(1-\frac{i-1}{kn}\right)^{t_2}\frac{i-1}{kn}\cdots\left(1-\frac{i-l+1}{kn}\right)^{t_l}\frac{i-l+1}{kn}\cdots \\ \cdots\left(1-\frac{i-j+1}{kn}\right)^{t_j}\frac{i-j+1}{kn}\cdot\left(1-\frac{i-j}{kn}\right)^{n-j-\sum t_l}$$

Summing over all possible ways of splitting n into j intervals yields formula (5.11) on the preceding page.

Limit Probability Distribution

Unfortunately, the probabilities are too complex to find a closed form solution. Note that there is hope for one though. The limit probability of state kn is zero. This state corresponds to a perfect star, all edges pointing to the one node, this state is impossible to reach. If that was to be the case, then the next swap would be guaranteed to be with the node at the center of the star, which would lose one of its edges, thus breaking the star.

The limit probability of state $kn-2$ can be easily expressed as a function of that of state $kn-1$ from the probabilities (5.11) on the preceding page. Consider the equilibrium between state $kn-2$ and $kn-1$. We have

$$P_{kn-2}\Pr[kn-2\rightarrow kn-1] = P_{kn-1}(1-\Pr[kn-1\rightarrow kn-1])$$

yielding

$$P_{kn-2} = kn(kn^{kn-1}-1)P_{kn-1}$$

From there, P_{kn-3} can be computed. In fact, since there is a single transition from any state to a higher numbered state, the limit probability distribution of

state i can be expressed in terms of all the higher states, yielding a theoretical closed form expression.

We resorted to numerical simulations and found that the standard deviation of the limit probability was $k/2$. This is a result of significant importance because it shows that one cannot interchangeably consider the synchronous and the asynchronous model. Simulations of an implementation of the swap protocol also yield the same standard deviation for the in-degree: k in the asynchronous version and $k/2$ in the synchronous version.

5.5 Conclusion

The distribution of the node's in-degrees is governed by the way the edges are created, copied and destroyed. The main difference between the normal protocol and the swap protocol is the speed at which the creation and destruction of edges occur. More reinforcement, that is, higher speed, decreases the standard deviation of the nodes' in-degrees. A large deviation is a practical issue because some of the nodes will have a large in-degree, making them bear a significant load. In the normal protocol the standard deviation can be large, but can be decreased by increasing the fanout. Switching to the swap protocol significantly decreases the standard deviation and lowers it to a level entirely compatible with practical applications. However, the swap protocol does not mix much and this may be an issue for some applications.

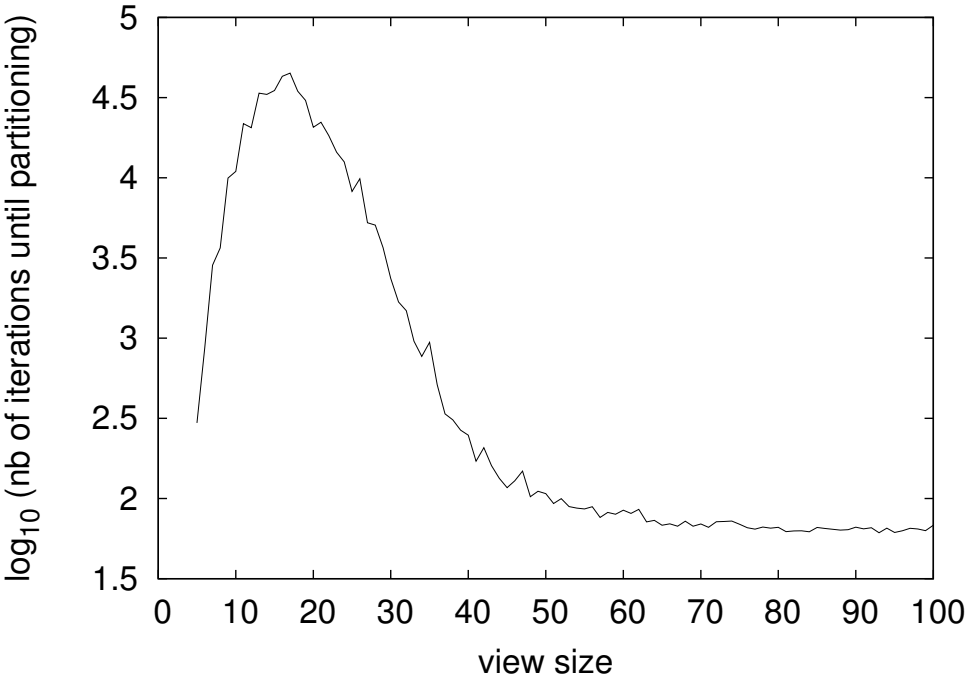
Looking at the concatenation of all the views is a powerful analytical tool to study the in-degree distribution of the nodes. There are two sources of variability in the in-degree distribution: the copying (some edges may be copied more often than others) and how long edges linger in the system. The large variability in

the in-degree distribution is mostly caused by the random nature of the process in charge of eliminating old edges from the views. Ideally, a pointer (an edge) created at time $t = 0$ would be moved around but not dropped or duplicated much, and deleted after some number of iterations l. In our protocol, a random process is in charge of removing the pointers. They are removed after an expected l iterations, but with high variability. This is the main source of variability in the in-degrees. Decreasing this variability sharpens the in-degree distribution. This can be achieved by increasing the reinforcement since this increases the rate at which edges are created and deleted, in effect decreasing the lifespan thus its variability. Increasing reinforcement means increasing the fanout in the main protocol or using the swap protocol, which has more reinforcement built-in. Alternatively, the lifespan variability can be decreased by keeping an age counter for every edge, making sure to remove the oldest edges first. This, even done locally, ensures that on the whole, the oldest edges are removed, significantly decreasing the lifespan variability. Cyclon [80] is a variant of our main protocol that includes an edge age counter. The in-degree distribution is very sharp, as shown by the authors' simulations. This is quite understandable from our theoretical analysis.

Chapter 6

Churn Analysis

Simulations exhibited the surprising behavior that increasing the view size decreased the time until partitioning, the time until one or more nodes gets disconnected from the main component of the network. A larger view size does not improve reliability. On the contrary, increasing the view size beyond some optimum leads to decreased connectivity. In Figure 6.1 we plotted the number of iterations until partitioning as a function of the view size.



The number of iterations until at least one node is disconnected from the main graph component is plotted as a function of the view size. There were 256 nodes in the graph. The churn rate was $1/32$: eight nodes were killed per round. The value for each view size is an average over 30 runs.

Figure 6.1: Partitioning with Churn as a Function of the View Size

Furthermore, other simulations suggest that the optimum view size is a constant, independent of the number of nodes in the system. However, view sizes need to be at least logarithmic in the system size for connectivity reason [29]. View sizes cannot grow logarithmically and at the same time be bounded by a constant. This is a serious scalability issue and will be explored in this chapter.

6.1 Churn Model

6.1.1 Other Model

A typical model is the one used in [67] and [52]. The nodes arrive according to a Poisson process with rate α and depart with an exponential distribution with rate parameter β. In steady state the number of nodes in the system is $N = \alpha/\beta$. The expected number of arrivals and the expected number of departure per time step is α.

6.1.2 Our Model

Here we study a different, slightly simpler model. Keeping the number of nodes in the system constant was a requirement in devising our model. In each round a number μ of nodes die and the same number μ of new nodes join the network. Each node has a constant probability p_{fail} of failing per round. Assuming that nodes fail independently, the number of failures per round is a binomial distribution of parameters n and p_{fail}. For an individual node, this corresponds to a geometric distribution with parameter p_{fail}. If the exponential distribution function is $\beta e^{-\beta t}$ then we have $\beta = p_{\text{fail}}$. The expected value of μ is

$$E[\mu] = p_{\text{fail}} n$$

Each time a node dies a new node immediately joins the network to replace the dead node. The new node may take over the view from the deceased node it replaces or compute a fresh view from the node(s) it bootstraps from. In either case the new view may contain dangling pointers. There may be some dead nodes in the view of the deceased node that the new node is assuming. There may also be dead nodes in the view of the bootstrapped node, or any other view the new node decided to request. We assume that nodes do not test their view entries for liveness.

6.1.3 Related Work on Churn

Some work has be done with respect to churn in a distributed system. Routing in a generic setting in the presence of churn was studied in [2]. Reference [50] is a theoretical statistical study of the Chord peer-to-peer protocol while simulations of various peer-to-peer schemes are compared in [51]. Reference [52], closest in spirit to our work, proves that the number of membership updates (joins or leaves) per unit of time a nodes receives needs to scale logarithmically in the system size.

6.2 Relation Between View Size and Fanout

6.2.1 Global State

Consider the concatenation $\mathcal{V}$ of the views as we did in Chapter 5. We evaluate the average number of dangling pointers in $\mathcal{V}$. Let k be the view size, n the system size and d the average number of live edges per view. The average node in-degree is d. The fraction of dangling edges is $^{(k-d)}/_{d}$. The average number of nodes dying per round is $p_{\text{fail}}n$.

Denote by f the fanout. At every iteration, each node contacts f nodes for reinforcement purposes.At every iteration, fn fresh, correct, edges are created and fn old edges are dropped. Running the protocols as described in Chapter 2 may lead to slightly less than fn fresh edges created at each round. This does not matter here.

In steady state, consider the number of correct and dangling edges being created in $\mathcal{V}$ at every iteration. We assume that mixing does not affect the number of live and dangling edges although it does in reality. Since nodes do not check for the liveness of edges before including them in their view (except maybe when selecting an edge to pull from, see note in 6.2.4 on page 97), the mixing does not change the average number of live or dangling edges. However, it does increase its variability, which we do not consider here.

6.2.2 Result

The number of edges being dropped by reinforcement is fn. Dropping an edge is the only way to replace a dangling edge by a live one with our assumptions. Out of these fn discarded edges, a fraction $(k-d)/k$ are dangling. On average, at every round, there are $fn(k-d)/k$ dangling edges being replaced by live ones. Since $p_{\text{fail}}n$ nodes die at every round, we create on average $p_{\text{fail}}nd$ dangling edges per round. In steady state, the number of dangling edges created or removed are equal. We have

$$\frac{k-d}{k} fn = p_{\text{fail}} dn \tag{6.1}$$

This yields a relation between the fanout and the average number of live edges per node:

$$f = \frac{k}{k-d} d p_{\text{fail}} \tag{6.2}$$

Even for a very conservative value of $d = k/2$ the fanout is linearly linked to d:

$$f = p_{\text{fail}} d$$

6.2.3 Interpretation

At every round, each node initiates f communications and on average is contacted by f other nodes. The fanout, f, is a direct measure of the amount of communications in the system. Communication is also the most important cost for a distributed system nowadays since both computation and reasonably sized storage are cheap.

The above results proves that the fanout needs to increase linearly in d, the average number of correct nodes per view. Keeping a large view size is of no use when most of it is incorrect. This result applies directly to our algorithm, as well as Newscast [42] and Cyclon [80].

Other algorithms like LPBCast [22] Scamp [29] have a separate communication channel to announce joins and graceful leaves. The "join" part has no effect on our result. Only a very aggressive "join" algorithm where the joining node communicates with k nodes in one[1] round might break our result. On the other hand, the "leave" part is important because nodes can elect for replacement certified dead nodes instead of randomly chosen ones. This is explained in detail in the following Section "Improved Algorithm." Note however that the graceful leave mechanism of Scamp and LPBCast requires nodes to stay significantly longer in the network (longer than their expected lifetime in our churn model) in order to leave gracefully. It also requires all nodes to leave gracefully. If the model allows nodes to crash with a fixed probability per node, then our result applies, regardless of the

[1] Or constantly many.

graceful leave mechanism.

6.2.4 Improved Algorithm

When a node u selects a node v from its view to pull from, it may happen that v is dead. We assumed, without making it explicit, that node u will then select other nodes to try to pull from, until it found a live node to pull from. This strategy ensures that node u will pull from exactly f live nodes, equivalent to having node u being reinforced f times. We ignore the fact that if node u is unlucky, it may take it a long time until node u finds f live nodes in its view, thus delaying the end of the round.

An improvement on the algorithm is the following: node u marks for replacement all the nodes it unsuccessfully tried to pull from. Some of these nodes might still be alive and be unreachable for some unknown reason, but we conservatively assume them to be dead. These marked nodes are then replaced in priority when node u updates its view. The number of dead nodes being dropped in the network increases since we replace dead nodes with hopefully live ones. Some simple algebra shows that the average number of nodes marked dead is $fn^{k-d}/_{d}$. These dangling edges get replaced by other nodes (from the views received by u) whose proportion of live nodes is the same as in the whole network. On average a fraction $^{d}/_{k}$ of these dangling edges are changed into live ones. This makes for $fn^{k-d}/_{k}$ less dangling edges. There are also the $fn^{k-d}/_{k}$ dead edges removed by the reinforcement process. In total $2fn^{k-d}/_{k}$ dangling edges are removed from the network in one round. The number of dangling edges created per round is unchanged: $p_{\text{fail}}nd$. In steady state we have:

$$2\frac{k-d}{k}fn = p_{\text{fail}}dn$$

which yields

$$f = \frac{1}{2}\frac{k}{k-d}dp_{\text{fail}} \tag{6.3}$$

This is an improvement of a factor 2 (compare with Equation (6.2) on page 95). However, this does not change the fact that the fanout needs to grow linearly in the number of correct nodes per view.

6.3 Oracle Guessing

Testing for the liveness of nodes present in one's view will break this result. Liveness testing is equivalent to assuming perfect guessing: when selecting a node at random to merge in one's view, always select a currently live edge and when selecting a node to discard, always select a dangling edge.

Assume that at every iteration a node u learns about k new nodes in the system. All of these k nodes were alive and present in the network at time t when they were sent to u. They may die anytime later, including time $t+1$ when u would first be able to use them. The mechanism ensuring that all sent nodes are live is of no interest here.

6.3.1 Probability of a Node Getting Disconnected

From [62]. Consider the entire knowledge that node u potentially has, not just the edges it selected to keep. This knowledge consists of the k nodes it has learned in round j for $0 \leq j \leq t-1$. For each round j, the probability that no node learned during that round is still alive at time t is $(1-(1-p_{\text{fail}})^{t-j})^k$. The probability that

all the nodes node u knows about are dead is:

$$\begin{aligned} P &= \prod_{j=0}^{j=t-1} \left(1-(1-p_{\text{fail}})^{t-j}\right)^k \\ P &= \left((1-(1-p_{\text{fail}})^1)\cdots(1-(1-p_{\text{fail}})^j)\cdots(1-(1-p_{\text{fail}}))^t\right)^k \end{aligned}$$

Denote by $p_{\text{live}} = 1 - p_{\text{fail}}$ the probability of a node not dying during a round. Using the fact that $(1-(1-p_{\text{fail}})^j) = 1 - p_{\text{live}}^j \geq (1-p_{\text{live}})^j$ we have

$$P = \prod_{j=1}^{j=t}(1-p_{\text{live}}^j)^k \tag{6.4}$$

$$P \geq \prod_{j=1}^{j=t}(1-p_{\text{live}})^{jk}$$

$$P \geq (1-p_{\text{live}})^{kt(t+1)/2} \tag{6.5}$$

Consider small values of t in Equation (6.5). The value of k needs to be at least logarithmic in the system size for the above probability to be sufficiently small. If k is not logarithmic but say constant then a linear fraction of the nodes would get disconnected.

$$k = \Omega(\frac{1}{p_{\text{live}}}\ln n) = \Omega(\ln n)$$

In the other direction, we can trivially upper bound the probability P of a node not knowing any live node from Equation (6.4) by upper bounding all terms but the first one by 1. We get

$$P \leq (1-p_{\text{fail}})^k \tag{6.6}$$

For $k \geq (\frac{c}{p_{\text{live}}}\ln n)$, Equation (6.6) yields

$$P \leq \left(\frac{1}{n}\right)^c \tag{6.7}$$

This bound is sufficiently small to prove that no node will be in the position of being disconnected for not having a single live node left among all the nodes it learned about until significantly many iterations of the protocol.

6.3.2 Interpretation

We computed the probability that all potential nodes that a node knows about have died. In such a case the node has no choice but to be disconnected from the rest of the network. This corresponds to the result of [52] claiming a $\Theta(\ln n)$ bound on the necessary and sufficient amount of membership update information a node needs to receive in order to stay connected.

Gossip-based membership systems as we described them in Chapter 2 provide sufficient information per round as long as the view size $k = \Omega(\ln n)$. However there are two serious practical issues. The first one is how to provide k *live* nodes to every node at every round. The difficulty is not in the number since every node stores k nodes but in how to ensure that these nodes are alive. Pinging every node in one's view is not desirable because of the associated costs and because it contradicts somewhat the philosophy of gossip algorithms.

The second issue is that nodes have to choose wisely which nodes to keep in their views. To keep only information of size $\mathcal{O}(k)$ locally, a lot of the nodes previously learned need to be discarded. An appropriate choice ensuring connectedness exists. However, this choice needs to be made a posteriori: nodes that will die in the near future should not be kept, nodes that will only die in the distant future should be kept.

Conclusion

Our result shows the following. In the absence of failure detectors, the simplest of which (but impractical) is to ping each node that is considered for inclusion[2] in a

[2]This also applies to nodes present in the previous round view being considered for inclusion in the current round view.

view, not only does the amount of membership information received at every node need to be at least logarithmic in the system size, the frequency of communications needs to scale linearly in the view size.

Chapter 7

View Randomness

One would like the elements of the views to be selected uniformly at random from the set of all nodes in the system. A view is said to be a uniformly random view if it has this property. In our simulations as well as those in [22, 29, 80, 41], the views are not uniformly random. Furthermore, our in-degree analysis of Chapter 5 proves that views cannot be uniformly random. However, although the views are not uniformly random views, the elements appear to have a high degree of randomness.

In this chapter we present some arguments supporting the fact that view elements are highly random. No formal claims are made. We merely present arguments in support of the experimental fact that views appear to be random. The arguments apply to all protocols mentioned in this work. The content of a view is rapidly dispersed and it is likely that the state of the system after $\mathcal{O}(\ln n)$ iterations is independent of the initial state. This suggests a "memory-less" property of the system: independence from close past.

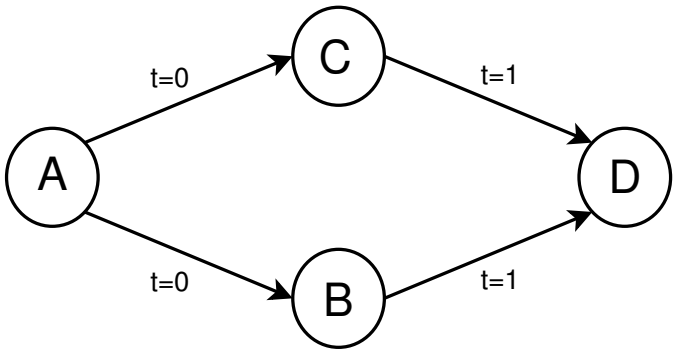
For reference, some strong results are presented in [16, 60], but in a system that does not reassemble much our protocols.

7.1 Collisions

Until otherwise mentioned, reinforcement is neglected.

7.1.1 Simple Push Algorithms

Consider Picture 7.1. We consider a push type of algorithm: node A selects a



An arrow denotes a communication at the time mentioned.

Figure 7.1: Collision in the Simple Push Algorithm

few nodes from its view to which it sends its view. Let B, C, D, E and F be nodes that are in A's views. In the figure, node A sends (pushes) its views $\mathcal{V}_A$ to nodes B and C. Nodes B and C compute their new view from what has been pushed to them. Assume that both nodes B and C decided to select D, E and F for their new view. Further, in the next iteration, both nodes B and C decide to communicate with node D. Node D will receive from both nodes B and C information about the existence of nodes E and F. This is redundant information. It would have been better if this information had been disseminated to two different nodes. Furthermore, this information about E and F is not traveling fast. Node A could have elected to communicate with node D directly instead of B and C. Node D would then have learned a round earlier about nodes E and F.

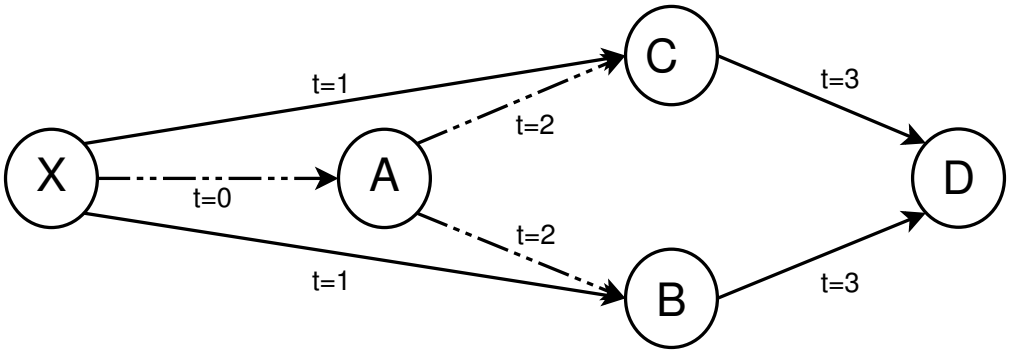
7.1.2 Delayed Push Algorithm

We considerer a variant of the push algorithm explained above in which each node stores two views, an information view $\mathcal{VI}$ and a sender view $\mathcal{VS}$. A node transmits its information view $\mathcal{VI}$ to nodes selected from its sender view $\mathcal{VS}$. Then

the information view becomes the sender view in the next iteration. The views are updated in the following way. For every node u:

$$\begin{cases} \mathcal{VS}_u(t+1) & \leftarrow & \mathcal{VS}_u(t) \\ \mathcal{VI}_u(t+1) & \leftarrow & \text{SELECT from } \cup_{i \in \{\text{Nodes that communicated with } u \text{ at } t\}} \mathcal{VI}_i(t) \end{cases}$$

The information view from the previous round is used to select the current communication targets.



An arrow denotes a communication at the mentioned time.

Figure 7.2: Collision in the Delayed Push Algorithm

Consider Figure 7.2. At time $t = 0$, node X sends its information view $\mathcal{VI}_X(0)$ to node A. Denote by B and C two nodes from this view that node A chooses to include in its next information view. The information view becoming the sender view of the next iteration, we next have $B, C \in \mathcal{VS}_X(1)$ and $B, C \in \mathcal{VS}_A(2)$. Both node X at time $t = 1$ and node A at time $t = 2$ elect to communicate with nodes B and C (both of which are in their respective senders view). At round 1 node X sends its information view $\mathcal{VI}_X(1)$ to nodes B and C. Denote by D a node from $\mathcal{VI}_X(1)$ that both nodes B and C will include in their time $t = 2$ information view. Node D is automatically part of the time $t = 3$ sender views of nodes B and C. At round 2 node A sends its information view $\mathcal{VI}_A(2)$ to nodes B and C. Denote

by $\mathcal{CVI}$ the part that both node B and C chose to include in their own view. At round 3, both nodes B and C select node D as a communication partner and node D receives $\mathcal{CVI}$ twice.

In the figure (Figure 7.2) arrows denote communications with different arrow styles indicating disjoint sets of information. Information is generated by node X and sent out to various nodes. The recipient of each piece of information uses the information to select its own communication partner (which is located on the right side of the picture). Consider the plain arrows for example. Information $\mathcal{VI}_X(1)$ is sent by node X at time $t = 1$ to nodes B and C. At time[1] $t = 1 + 2 = 3$ nodes B and C use part of the information to select a communication target, node D. Now consider the dotted arrows: node A receives some information $\mathcal{VI}_X(0)$ at time $t = 0$ and uses parts of that information at time $t = 0 + 2 = 2$ to selects its communication partners (nodes B and C). The figure we described shows a case where the same information, $\mathcal{CVI} \subset \mathcal{VI}_A(2)$, is received twice at the same node (node D).

7.1.3 Other Algorithms

Variants of the delayed (push) algorithms are possible: instead of using the view from the previous round to select communication partners, use the view from the 2nd, 3rd or jth previous round. While we have not been able to construct an example where a collision occurs, we are convinced that such an example exists. However, the longer the delay between when the information is received and when it is used to draw communication partners, the less likely the collision is.

[1]The protocol imposes a two rounds delay between when the information is received by a node and when the node uses it to select communication partners.

Pull algorithms on the other hand do not exhibit this behavior: collisions are not possible. While it is possible for some information originating at one node to reach another node using two distinct paths, one has to rely on luck for this to happen. There is no systematic way of designing a communication sequence so that some information is sent to the same node twice.

In order for a collision to happen, some information that has been sent to nodes B and C needs to be received by a node D from both B and C. This is possible in a push system, not in a pull one. The reason is simple. In a push system, information ($\mathcal{VI}$) and routing ($\mathcal{VS}$) are correlated either directly or indirectly since the information affects the routing by the relation $VS(t+j) = \mathcal{VI}(t)$. The routing choices are made after learning the information. On the contrary, in a pull algorithm, the routing cannot be affected by the view information since the routing decisions are made *before* a node learns of the content of a view. A node learns some view information $\mathcal{VI}_A$ only after its routing duties regarding $\mathcal{VI}_A$ are over.

7.1.4 Conclusion

There are "collisions" in push algorithms, that is, information arriving at the same node by two distinct paths. This information would have been more efficiently disseminated had it been delivered to two different nodes instead of one. This suggests that the information is not mixed as well or as efficiently as it could be. Mixing time is the length of time it takes before the probability of the information reaching node u is approximatively the same for all nodes u. Collisions show that the mixing time is longer than what it could be.

On the other hand, pull algorithms do not have these collision issues, which is why we consider them to be a superior alternative. Note however that the

reinforcement is a push mechanism, and reinforcement is required, so the collision issues cannot be entirely avoided.

The concept of mixing time and convergence are used informally here. The underlying Markov Chain being reducible, the mixing time is not defined. We use "mixing time" to denote the time to converge to an almost uniform distribution.

7.2 View Element Mixing without Reinforcement

Define the connectivity graph $\mathcal{G}(t)$ to be a directed graph where the nodes are the same as the nodes in the system and the edges are given by the views at time t. If node v is present in u's view then the edge $(u \rightarrow v)$ is present in the graph. The connectivity graph changes at each protocol iteration. Moreover, the views are sent along the graph edges. The protocol view exchanges corresponds to the views doing a random walk on the connectivity graph. Consider k iterations of the protocol starting at time $t = 0$. These k iterations correspond to a random walk of exponential length (in k) over the connectivity graph $\mathcal{G}(0)$, as we will make clear in the following paragraphs.

7.2.1 Pull Algorithm

Consider $\mathcal{G} = \mathcal{G}(0)$, the connectivity graph at time $t = 0$, and let $\mathcal{G}$' be the graph $\mathcal{G}$ in which the direction of all the edges is reversed. Consider a view $\mathcal{VI}$. At each protocol iterations the information contained in $\mathcal{VI}$ will be broken in small pieces, some of these small pieces will be recombined with others, possibly being used multiple times, while the others will be dropped. Consider the view $\mathcal{VI}_A(0)$ held by some node A at time $t = 0$. Denote by $\mathcal{VI}_A$ some piece of $\mathcal{VI}_A(0)$ that is still present in the system after k iterations of the protocol. We will show that after k

protocol iterations, $\mathcal{VI}_A$ has traveled exactly $2^k - 1$ steps in $\mathcal{G}$', starting from A.

The proof is by induction. In the first protocol iteration, $\mathcal{VI}_A$ is pulled from A by some node B. Node B had node A in its view in order to pull node A, that is, there was an edge $(B \rightarrow A)$ in $\mathcal{G}$, or equivalently, an edge $(A \rightarrow B)$ in $\mathcal{G}$'. The information $\mathcal{VI}_A$ took a one-step walk in $\mathcal{G}$', from A to B.

Assume the content of views at iteration t is the aggregation of pieces of the original views that have traveled $2^t - 1$ steps in $\mathcal{G}'$. These views were held by some node at time $t = 0$. At time $t+1$, the piece of information $\mathcal{VI}_A$ that we consider is at some node X and is pulled by some node Y. By our induction hypothesis, $\mathcal{VI}_A$ had traveled $2^t - 1$ steps in $\mathcal{G}$' and now takes a step $(X \rightarrow Y)$ in the connectivity graph considered at time $t+1$. This graph however is not the same as $\mathcal{G}$, which was defined at time $t = 0$. The step $(X \rightarrow Y)$, a single edge in $\mathcal{G}(t+1)$, corresponds to a walk on many edges in $\mathcal{G}'$. Node Y had to have node X in its view at time t to be able to pull from X. This knowledge of X is a piece of information that originated at some node Z at time $t = 0$. Therefore there must be an edge $(Z \rightarrow X)$ in $\mathcal{G}$ and equivalently an edge $(X \rightarrow Z)$ in $\mathcal{G}$'. The knowledge of node X that node Z held at time $t = 0$ traveled $2^t - 1$ steps from node Z to node Y by our induction hypothesis. The complete path $\mathcal{VI}_A$ took is then $A \rightarrow X \rightarrow Z \rightarrow Y$. See Figure 7.3 for a graphical representation of the path. The information $\mathcal{VI}_A$ went from node A to node X in $2^t - 1$ steps by our induction hypothesis, then went from node X to node Z in one step and finally went from node Z to node Y in $2^t - 1$ steps, for a total of $(2^t - 1) + 1 + (2^t - 1) = 2^{t+1} - 1$ steps in $\mathcal{G}$'.

7.2.2 Push Algorithm

Similar results showing that k protocol iterations correspond to the original pieces of information taking an exponentially (in k) long random walk in the connectivity

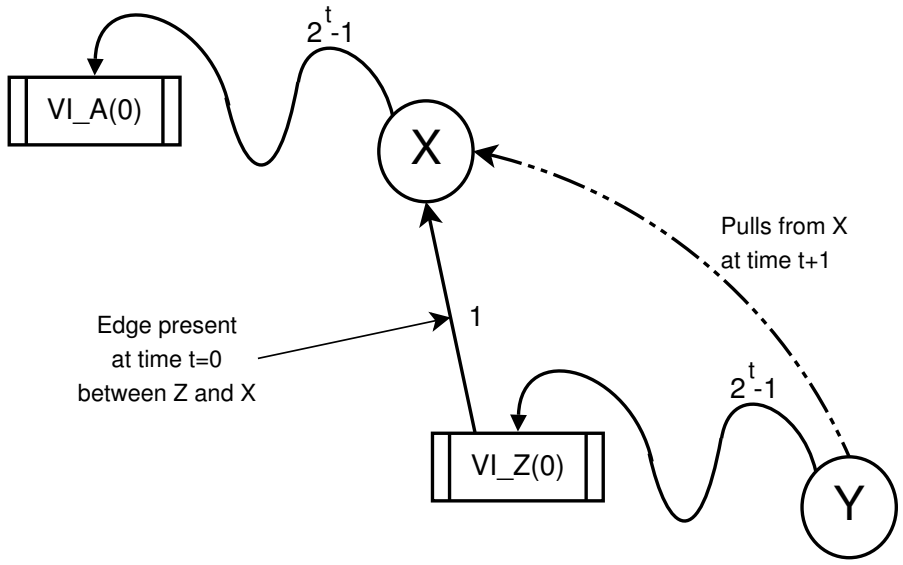


Figure 7.3: Travel of a View in the Pull Algorithm

graph are obtained for the other algorithms. Denote by I a step forward and by I^{-1} a step backward in the connectivity graph $\mathcal{G}_{\mathcal{VI}}$ defined by the information views $\mathcal{VI}$. Denote by S a step forward and S^{-1} a step backward in the connectivity graph $\mathcal{G}_{\mathcal{VS}}$ defined by the sender views $\mathcal{VS}$. Denote by U_i the sequence of steps in the graphs $\mathcal{G}_{\mathcal{VI}}$ and $\mathcal{G}_{\mathcal{VS}}$ that represents the path of a message originating at node X at time $t = 0$ and reaching some node Y at time $t = i$. The sequence U_i is a string of the four characters I, I^{-1}, S, S^{-1}. While U_i denotes a unique sequence of these four characters, it does not denote a unique path in the graphs, rather, it denotes any path in the graphs such that the steps conform to the sequence U_i. The length of the sequence U_i is equal to the number of steps taken in the graphs.

To illustrate, assume the sequence U_i is some pseudo driving indications. Instead of the usual four, there are many streets meeting at each intersection. I means "turn left" and I^{-1} means "turn right." Since there are many streets on

the left, I indicates to turn onto *some* street on the left. Which one is not known. A sequence U_i could be "left, right, left, left" and is not good driving directions as the sequence can lead to many different destinations, depending on which particular street is chosen at each intersection. Nevertheless, the sequence describes a well defined category of driving directions, the set of all the driving directions indicating a turn onto some street to the left, then a turn onto some street on the right, etc.

7.2.3 Delayed and Non-Delayed Push and Pull Algorithms

Delayed Push Algorithm

In the delayed push algorithm we have

$$\begin{aligned} U_1 &= S^* \\ U_2 &= S^* I^* \\ U_3 &= S^* I^* S^{-1*} I \\ U_4 &= S^* I^* S^{-1} I \ I^{-1*} S^{-1} I \\ &\dots \\ U_n &= U_{n-1}(U_{n-2})^{-1} I \end{aligned}$$

The superscript -1 indicates to reverse the steps. For example $(IU^{-1})^{-1} = UI^{-1}$. The superscript $*$ indicates a random choice which may lead to the (same) information being propagated to several destinations. For example "left*" indicates to turn onto some, possibly more than one, street on the left. Information can be replicated and can be sent on the second street as well as the third street on the left.

In the $U_n = U_{n-1}(U_{n-2})^{-1}I$ formula, the stars are on the elements of U_{n-1} that had a star and on the first element of U_{n-2}^{-1}. The stars are removed from the other elements of U_{n-2}^{-1}.

The length of the sequence, that is, the number of steps taken in the graphs is given by the Fibonacci sequence and as such is still exponential in the number of protocol iterations.

Pull Algorithm

Using the above notation, the pull algorithm is described by:

$$\begin{aligned} U_1 &= I^{-1*} \\ U_2 &= I^{-1*}I^{-1*}I^{-1} \\ U_n &= U_{n-1}I^{-1*}U_{n-1} \end{aligned}$$

In the formula giving U_n, the stars are kept on the left U_{n-1} and removed from the right U_{n-1}. Remember, both U_{n-1}'s denote the same sequence of backward steps in the graphs but do not indicate the same path in the graphs.

Delayed Pull Algorithm

The steps taken in the connectivity graphs are the following:

$$\begin{aligned} U_1 &= S^{-1} \\ U_2 &= S^{-1}I^{-1} \\ U_n &= U_{n-1}I^{-1}U_{n-2} \end{aligned}$$

Note that all steps are taken backward in the graphs. This was also in the case in the simple pull algorithm.

Simple Push Algorithm

The steps taken in the connectivity graph are the following:

$$\begin{aligned} U_1 &= S \\ U_2 &= SS^{-1}S \\ U_n &= U_{n-1}U_{n-1}^{-1}S \end{aligned}$$

The formula suggests that the walk has a length of $|U_t| = 2^t - 1$ steps. This is not exactly the case because the walk can backtrack, making for a traveled distance smaller than the number of steps taken. For example, consider the walk on Figure 7.1 on page 103 representing information about node E held by node A at time $t = 0$ sent to node D via node B (or C). From the picture, the path from A to E appears to be A→B →D→E. In reality, node B learned about node D from node A's view. The path really is A→B→A→E which is equivalent to A→E. The path is one step long, not three.

To the contrary of the previous algorithms, simple push is the only algorithm where k iterations of the protocol may not correspond to the original pieces of information taking an exponential long (in k) random walk on the connectivity graph.

7.3 View Element Mixing with Reinforcement

7.3.1 Pull without Reinforcement

Consider a node Y in some node A's view at time t and denote by P_t the sequence of steps describing the path from A and Y in the connectivity graph $\mathcal{G}$ (at time $t = 0$). Like previously, the sequence of steps does not represent any path between A and Y

but the path that represents the unique way the protocol manipulated information about the nodes to let node A learn about node Y. By abuse of language, we use the "path to reach a node Y in node A's view" to refer to this sequence P_t.

Remember that U_i denoted the path information originally stored at node X followed before reaching node A. Assume node Y is in node X's view at time 0. The path $X \rightsquigarrow A$ was denoted by U_i and the path $A \rightsquigarrow X \rightarrow Y$ is denoted by P_i. We have $P_i = (U_i)^{-1} I$.

Using the previous notation, in the pull algorithm we have

$$\begin{aligned} P_0 &= I \\ P_1 &= I^2 \\ P_i &= I^{2i} \end{aligned}$$

The proof is by induction. It takes one step forward in the connectivity graph $\mathcal{G}$ to reach a node in one's view at time $t = 0$. At time $t = 1$, a node B in u's view comes from the view of a node A that u pulled at time $t = 0$. It takes one step to reach node A from u and another one to reach node B.

Assume it takes 2^i steps in $\mathcal{G}$ to reach a node in one's view after i iterations of the protocol. Assume node A has node X in its view and node X has node Y in its view. Denote by $\mathcal{P}_{AX}$ the path from node A to node X (corresponding to the sequence P_i) and $\mathcal{P}_{XY}$ the path from node X to node Y in $\mathcal{G}$, also corresponding to the sequence P_i.

The view of node A at iteration $i + 1$ contains the views of nodes that node A pulled at iteration i. Consider a node Y presently in A's view that A got from pulling some node X. We evaluate the path from node A to node Y. This path is $\mathcal{P}_{AY} = \mathcal{P}_{AX}\ \mathcal{P}_{XY}$. We have $P_{i+1} = P_i P_i = I^{2(i+1)}$ by our induction hypothesis

since the two paths $\mathcal{P}_{AX}$ and $\mathcal{P}_{XY}$ have length 2^i. The path $\mathcal{P}_{AX}\ \mathcal{P}_{XY} = \mathcal{P}_{AY}$ connecting A to Y in $\mathcal{G}$ has a length of $2^i + 2^i = 2^{i+1}$.

7.3.2 Pull with Reinforcement

Reinforcement is now reintroduced. The difference with the previous paragraph resides in the content of node A's view at iteration $i+1$, $\mathcal{VI}_A(i+1)$. The view $\mathcal{VI}_A(i+1)$ contains nodes from pulled views and may also contain nodes that pulled A. As previously, denote by Y a generic node that node A included in its view after pulling Y from some node X. Formally: $X \in \mathcal{VI}_A(i)$, $Y \in \mathcal{VI}_X(i)$ and $Y \in \mathcal{VI}_A(i+1)$. Denote by Z a node issued from the reinforcement. Node Z had node A in its view at time $t = i$ and decided to pull node A.

The view of node A at time $t = i+1$ is constituted of some nodes like Y and some nodes like Z. The path from node A to node Y's is P_iP_i as explained in the previous paragraph. The path from node A to node Z is simply $(P_i)^{-1}$. Node Z had node A in its view at iteration $t = i$ so P_i denotes the sequence of steps from node Z to node A. The path from node A to node Z is the reverse, P_i^{-1}. We have $P_{i+1} = P_iP_i \mid P_i$ where the symbol "|" denotes the operator "or".

$$\begin{aligned} P_0 &= I \\ P_1 &= I^2 \mid I^{-1} \\ P_i &= P_{i-1}P_{i-1} \mid P_{i-1} \end{aligned}$$

Because reinforcement is unlikely to occur, the length of the sequence P_i is almost certainly exponential in i, thus the number of steps taken in the connectivity graph is still exponential in the number of protocol iterations (except in some extremely unlikely cases).

7.3.3 Interpretation and Conclusion

The elements forming a node's view after k protocol iterations come from an exponentially (in k) long random walk in the original connectivity graph. This suggests that view elements are highly random. However, no formal claims can be made due to the following two issues:

1. Contrary to random walks on undirected graphs, random walks on directed graphs can take a long time to mix. In the worst case the mixing time of a directed graph is exponential in the graph size[2] and there is no known technique to bound the mixing time of a directed graph. However, recent work by Fan Chung raises some hope. In [14] she proves that the mixing time of directed regular graphs as well as Eulerian graphs[3] is (a small) polynomial in the graph size. The connectivity graph $\mathcal{G}$ is neither regular nor Eulerian, however $\mathcal{G}$ is close to being regular, thanks to our in-degree bounds from Chapter 5, leading us to conjecture that the mixing time of the connectivity graph is polynomial in the system size.

2. Random walks for different elements of a same view as well as for elements of distinct views are potentially correlated: every random choice made by the algorithm affects several view elements. However the groupings of correlated elements are different for each random choice and the number of random choices is very large, making the weight of each correlation very small. We consider the correlation to be insignificant and the view elements to be random.

[2]The graph size is the number of nodes in the graph.
[3]A directed graph is Eulerian if for all node u, in-degree(u)=out-degree(u).

Despite these issues and since the number of steps taken in the connectivity graph is exponential in the number of protocol iterations, we believe the content of a view after $\mathcal{O}(\ln n)$ iterations to be highly random and in particular independent of the system's original state (assuming a good original state).

We conjecture the following memoryless property of the system. Assume the system not to be in a particularly bad state at time $t = 0$. The system state at time $t = \mathcal{O}(\ln n)$ is independent of what it was at time $t = 0$.

Chapter 8

Conclusion

In this thesis we have presented several results regarding gossip-based membership algorithms. These algorithms are unique among membership algorithms in that the membership information stored at the nodes changes over time, independent of actual membership changes.

Static membership algorithms react after each individual failure and attempt to recover from each of them. These algorithms almost always assume successive failures to be uncorrelated. A long succession of failures could drive a static membership algorithm into a non-optimal state. Individual members usually cannot detect that the system is in such a state. When the members can locally detect that the membership is in a bad shape, it often is the sign of a desperate situation, and too late for a recovery mechanism to succeed.

Dynamic membership algorithms avoid this issue by continuously converging towards a good equilibrium. There is no need to detect failures since the protocol is always evolving. Correlated failures, unlikely fault sequence, etc. will not prevent the protocol from evolving. If the protocol is correctly designed, the system will return to (a good) equilibrium.

A dynamic algorithm may seem more costly than a static algorithm. On the practical side, a dynamic (proactive) algorithm potentially sends many more messages than a static (reactive) algorithm, since a proactive algorithm communicates even in prolonged absences of membership changes. However, this is unlikely to happen in large systems. The more nodes there are, the more frequently the membership changes, and the more messages are generated by a reactive algorithm.

Thus, in a large system, a dynamic algorithm is not necessarily more expensive than a static one.

On the theoretical side, a dynamic protocol is much more difficult to understand and analyze. The crucial point is to characterize the state to which the algorithm converges (or tries to converge). Determining that equilibrium state, determining how fast the convergence is, and identifying cases under which the algorithm may fail to converge, can all be difficult questions to answer.

In this thesis we considered an example of a dynamic algorithm, a gossip-based membership algorithm, and provided partial answers to the questions mentioned above. The overall work offers sufficient guarantees for an application-level gossip broadcast (and potentially other applications) to function satisfactorily. It also raises a number of new questions.

Convergence and Non-Partitioning In Chapter 3 we presented a simple yet powerful model for analyzing the behavior of gossip-based membership algorithms. Consider a partition A-B of the network graph. The fraction of edges pointing to nodes in A is an estimate of the fraction of A nodes. Let x be the fraction of edges from nodes in A that go to A nodes and y the fraction of edges from nodes in B nodes that go to A nodes. The values x and y are the estimates respectively by set A and set B of the fraction of A nodes. Studying the evolution of these estimates is sufficient to predict the overall behavior of the membership algorithm. In particular the analysis shows that any significant lack of edges across a given network cut will be corrected extremely rapidly. The analysis also shows that the network diameter is small.

Our analysis was applied to the two gossip-based protocols described in Chap-

ter 2. The analysis is simple and powerful enough to apply to other gossip-based membership protocols as well. We believe that it could be extended to practically any dynamic membership algorithm that works by maintaining per-node views. The size estimates that we defined are graph properties. Their evolution captures the evolution of the graph. The equations themselves and their analysis are specific to the membership protocol under consideration, but it should always be possible to formulate such equations. Even if a theoretical analysis of these equations is impractical, numerical evaluation can be used to investigate the overall behavior of the protocol and the evolution of the number of edges across a cut.

An analysis similar to the one described in Chapter 3 could be applied to peer-to-peer systems providing routing capabilities (like Chord, Viceroy, Pastry, Tapestry and others) to evaluate how these systems operate under churn. These systems do not keep membership views *per se*. However they keep routing tables. Applying the analysis considering the routing tables instead of the membership views should lead to interesting results. The analysis would give the evolution of the number of edges across a given network cut. A decrease in this number will be reflected by a decrease in the number of known routes across the cut and presumably signal increased usage, possibly saturation, of the remaining routes. If the number of edges across the cut decreases even more, there is a risk not only of saturated routes but also of network partition.

In Chapter 4 we considered a simplified model of a gossip-based membership algorithm, proving that the probability of a network partition forming is exponentially small even in the face of heavy churn. More precisely, we showed that the expected time before the network partitions is exponentially small in both the size of the departing set and the size of the views. Thus, for a sufficiently large system

with moderately large views, the system will *never* partition. The only potentially disconnected components will be isolated nodes that only have dangling edges left in their views.

Node Properties The load of a node is directly correlated to its in-degree. Having a relatively tight in-degree distribution is a necessity when the goal is to share the load evenly among all nodes. An in-degree analysis seems to be highly protocol dependent. We provide some bounds for both of the protocols we studied in Chapter 5. These analyses differ from that of LPBCast or Scamp. No analysis we are aware of applies to Cyclon, though our reinforcement concept and our "union of the views" help explain why this protocol exhibits a very sharp in-degree distribution.

In all these algorithms the in-degree distribution is the same for every node and all nodes have the same in-degree in expected value. However, the in-degree can be individually adjusted with some minor modifications to the protocols. If a node works twice as fast as the other nodes in the network, then its in-degree will on average be twice that of the other nodes. By adjusting the speed at which each node functions, almost any degree distribution of the expected in-degree can be obtained. When adjusting the speed, it is only necessary to adjust the speed of the reinforcement part. Note, however, that we have not investigated the other possible consequences of running different nodes at different speeds. For example, the effect on reliability is unknown.

In Chapter 7 we gave arguments regarding the quality of the randomness of the view content. These arguments are protocol independent. While non-rigorous, they strongly support the experimental evidence showing that the view elements

have a high degree of randomness. Applications using the randomness of the view elements as source of randomness are potentially numerous. The first applications to come to mind are gossip application-level broadcasts like Araneola [64], but many other applications could profit from the ability to select peers at random. These include applications using the aggregation techniques described in [47] and applications like T-Man [43] explicitly designed to use the randomness of the view elements provided by the membership management.

Further work None of the gossip-based membership algorithms we have discussed provide views that truly are uniform random samples of the membership, and we do not anticipate future algorithms that will do so. However, we expect further research to characterize more precisely the randomness of and correlation between the view elements. Particularly useful results would prove that the randomness provided by such membership management is sufficient for specific algorithms to function with good properties.

Another direction for further work is to approximate a different, non-uniform, distribution on the view elements. For example the elements in node u's view could be distributed according to an inverse power law in the distance between the element and node u. The further away node v is from node u, the less likely node v will be in node u's view. Inverse power law distributions have been shown to have good properties. For example, a message propagated using a gossip algorithm will radiate out of the originating node instead of reaching nodes in a disordered way. This property is useful in situations like propagating a fire alert, when the nodes closest to an event need to be notified first. There are other problems that can be solved using an inverse power law distribution but not a uniform distribution [48].

Scalable membership services providing views distributed according to an inverse power law distribution have yet to be developed.

Churn Churn is a characteristic of any large distributed system. Each node has an individual probability of failure. Assuming nodes fail independently, the number of nodes failing per time unit scales linearly with the number of nodes in the system. The maintenance cost born by the system is large because the cost of repairing a single failure grows with the system size. Often the cost is linear, making the overall cost scale with the square of the system size.

A typical solution to this problem is to exploit randomized algorithms that tolerate systems in which failures have not been completely repaired. This allows for a gradual approach to recoveries, significantly reducing their (amortized) cost. Gossip-based membership algorithms fall into this category: there is no attempt to discover failed nodes, but the failed nodes are eventually purged from the system.

A churn linear in the system size has consequences of its own. Members require frequent membership updates or face the certainty of getting disconnected for only knowing members that have left the network. In any time frame, if m is the number of membership changes occurring, each network node needs to receive notification of at least $\Omega(\ln m)$ of these membership changes in order to stay connected. In practice it may be difficult to provide such updates. A node may have died by the time the information announcing its joining reaches the last nodes in the graph, making the join announcement incorrect. Similarly, a node may crash without sending any message announcing that it is leaving the network. Thus, membership updates reliably announcing network departures will be difficult to provide.

An alternative is to consider algorithms where nodes communicate their belief

of the current membership. Nodes may not know of all nodes in the system and may think that departed members are still present in the network. Membership algorithms that store only partial information about the membership fall in this category. We proved in the case of our gossip-based membership algorithms that the number of communications per time unit had to scale with the size of the information kept at every node. This result has an unintended consequence: keeping the number of communications constant and increasing the amount of information stored at every node is counter productive as the increased amount of information is not kept current. More information is available at each node, but the information is more likely to be out-of-date (and useless), so the overall effect is to degrade performance.

This result is likely to apply to many systems, not just gossip-based membership systems. If so, the result is a serious argument in favor of limiting the amount of information stored at every node. Scalability goals led to an important body of work in designing distributed systems that could scale to billions of nodes by storing only a logarithmic amount of information at every node. Logarithmic information is very small and recent work noted that storing an amount of information proportional to a small power of the system size was entirely manageable [34], leading to an increase in performance. These performance analyses did not take into account the cost of communications needed to maintain stored information accuracy. Our result suggests that communication costs scale with the size of the locally stored information and that communication costs are a factor limiting scalability that needs to be considered.

Appendix A

Simulation Results

Numerous simulations of the original algorithm were run. There results are summarized in tables A.1 to A.3 on pages 126–128.

There were some implementation choices left open in the the description of the protocol in Chapter 2. Below we detail the quantities we set and the ones we chose to variate, as well as some design decisions.

Fixed Quantities:

Reinforcement The reinforcement weight was set to $\omega = 100,000$, approximating an infinite value and ensuring that all reinforcement nodes where selected when possible.

Fanout The fanout was set to 2. A value of 1 denotes a special case where no mixing occurs. A fanout value of 3 or more significantly increases the time until partitioning, making a partition virtually impossible to observe.

Variable Quantities:

View size Several view sizes were tested, from 4 to 12. View sizes larger than 12 resulted in excessive running time and could not be tested. The process is memory bound, the limiting factor is the large number of random memory access.

Churn The number of killed nodes per round, churn, was varied. When the churn value μ is less than one, a node was killed with probability μ. When the value is more than one, exactly μ distinct nodes were killed.

View initialization The view of a joining member was either initialized to that of the node it was replacing, that of a random live node, or that of node 0.

Design choices: We made the following choices:

Duplicates removal Views do not contain duplicates. Remember the list $\mathcal{L}_1$ comprising the views of the pulled nodes from Section 2.2.1 on page 16. We build this list so that it does not contain duplicates. Furthermore, if a node is present in $\mathcal{L}_2$, it is not included in $\mathcal{L}_1$. The new view of a node is computed by selecting k different elements from $\mathcal{L}_1$ or $\mathcal{L}_2$. These k elements will be distinct.

Pulling from a dangling edge A node u selects f nodes to pull from. If some of these nodes are dead, node u does not get to select other nodes to pull from. In the case where all the nodes node u tried to pull from are dead, node u keeps its old view for the next round, replacing some of the entries by the nodes that pulled u, if any.

Dying and joining When a node dies, a new node immediately replaces it in the network, thus keeping the total number of nodes constant. Nodes do not get reinforced in their first round.

Table A.1: Number of Iterations until Partitioning, View Size Is 4

Reinforcement is 1000000 Fanout = 2 View Size = 4

Nb iterations till partition		Partition Size																	
16	Use Old																		
#killed/rd	8		4		2		1		0.8		0.6		0.4		0.2		0		
Average	**2**	**2.40**	**3**	**2.40**	**7**	**1.87**	**44**	**2.27**	**147**	**2.03**	**237**	**1.67**	**527**	**1.83**	**2,663**	**2.13**	**266,940**	**6.10**
Std	0	2.71	1	2.04	3	1.72	36	1.64	82	1.79	147	1.42	442	1.09	2,876	1.38	294,186	0.99
Max		7		8		8		7		7		7		5		7		8
Min		0		0		0		1		0		0		0		1		4
16	Always 0																		
	8		4		2		1		0.8		0.6		0.4		0.2		0		
Average	**2**	**1.23**	**4**	**2.13**	**8**	**1.57**	**49**	**1.43**	**557**	**0.70**	**1,333**	**1.00**	**2,477**	**0.97**	**10,857**	**1.10**	**213,750**	**5.97**
Std	0	1.55	1	2.03	4	1.28	39	1.01	475	0.65	1,576	0.95	2,950	0.72	10,956	0.40	249,354	1.22
Max		6		8		5		5		2		3		3		2		8
Min		0		0		0		1		0		0		0		0		4
16	Random Copy																		
	8		4		2		1		0.8		0.6		0.4		0.2		0		
Average	**2**	**2.60**	**3**	**2.23**	**7**	**1.17**	**55**	**1.90**	**803**	**1.47**	**1,237**	**1.50**	**3,677**	**1.70**	**13,223**	**1.37**	**291,170**	**6.30**
Std	0	2.69	1	1.65	4	0.70	51	1.49	877	1.36	1,365	1.53	2,978	1.39	12,798	0.81	201,631	1.32
Max		8		6		3		8		7		8		7		4		8
Min		0		0		0		1		0		0		0		0		4
64	UseOld																		
#killed/rd	8		4		2		1		0.8		0.6		0.4		0.2		0		
Average	**4**	**2.00**	**18**	**1.30**	**377**	**1.27**	**3,703**	**1.80**	**7,913**	**1.83**	**10,287**	**1.67**	**25,237**	**1.60**	**78,887**	**1.93**	**3,024,070**	**5.20**
Std	2	1.70	15	0.65	459	0.52	3,514	0.81	5,612	1.15	9,502	0.80	27,508	0.77	67,341	1.23	2,034,327	1.47
Max		9		3		3		4		4		3		4		6		11
Min		0		0		1		1		0		0		1		1		4
64	Always 0																		
	8		4		2		1		0.8		0.6		0.4		0.2		0		
Average	**4**	**1.73**	**29**	**1.10**	**244**	**1.33**	**3,923**	**1.60**	**22,090**	**1.40**	**26,597**	**1.17**	**53,677**	**1.63**	**113,690**	**2.00**	**3,112,387**	**5.53**
Std	2	1.01	24	0.40	258	0.61	3,083	0.97	20,973	0.72	27,256	0.38	53,548	0.96	79,274	1.46	2,983,131	1.01
Max		4		2		3		5		4		2		5		7		7
Min		1		0		1		1		0		1		1		1		4
64	Random Copy																		
	8		4		2		1		0.8		0.6		0.4		0.2		0		
Average	**5**	**1.80**	**21**	**1.30**	**303**	**1.30**	**3,460**	**1.47**	**19,943**	**1.13**	**50,023**	**1.50**	**85,047**	**1.40**	**171,440**	**1.57**	**3,676,400**	**5.47**
Std	1	1.13	14	0.75	245	0.70	3,394	1.01	24,645	0.35	54,652	0.73	77,534	0.72	148,851	0.68	3,764,894	1.53
Max		5		4		4		5		2		3		4		3		10
Min		1		1		1		0		1		1		1		1		4
256	UeOld																		
#killed/rd	8		4		2		1		0.8		0.6		0.4		0.2		0		
Average	**77**	**1.20**	**1,511**	**1.20**	**15,242**	**1.57**	**72,360**	**1.57**	**98,670**	**1.50**	**188,433**	**2.10**	**394,310**	**2.13**	**802,540**	**2.27**	**8,959,893**	**5.20**
Std	79	0.48	1,179	0.41	17,541	0.77	51,981	0.82	95,158	0.68	164,385	1.21	290,499	0.97	490,930	1.46	9,582,112	1.21
Max	384	3	3,677	2	74,840	4	223,907	4	449,500	3	754,200	6	1,391,400	5	1,879,800	6	37,598,800	9
Min	5	1	50	1	342	1	2,401	1	16,300	1	8,900	1	28,000	1	10,900	1	54,400	4
256	Always 0																		
	8		4		2		1		0.8		0.6		0.4		0.2		0		
Average	**89**	**1.10**	**1,388**	**1.33**	**8,842**	**1.50**	**78,811**	**1.70**	**117,757**	**1.80**	**272,060**	**1.60**	**466,853**	**1.87**	**1,197,183**	**1.87**	**7,116,887**	**4.87**
Std	81	0.31	1,159	0.61	7,474	0.86	77,032	1.06	116,163	1.03	242,770	0.72	443,075	0.73	1,338,785	1.11	5,309,459	0.94
Max	327	2	3,839	3	31,921	5	342,915	5	484,600	5	918,400	3	2,042,900	4	6,959,500	6	21,248,500	8
Min	3	1	98	1	24	1	579	1	3,300	1	500	1	6,000	1	149,400	1	25,500	4
256	Random Copy																		
	8		4		2		1		0.8		0.6		0.4		0.2		0		
Average	**105**	**1.07**	**1,029**	**1.27**	**12,362**	**1.43**	**73,796**	**1.73**	**159,100**	**1.57**	**329,203**	**1.73**	**396,430**	**1.73**	**1,034,117**	**2.27**	**7,374,823**	**4.73**
Std	96	0.25	855	0.52	10,497	0.68	57,639	0.83	176,173	0.86	332,575	0.78	416,295	0.83	1,149,643	1.23	6,604,101	0.87
Max	477	2	2,972	3	42,012	3	216,436	4	850,400	5	1,164,000	3	1,561,200	4	4,812,400	6	27,803,500	7
Min	10	1	7	1	290	1	1,541	1	1,300	0	4,900	1	100	1	12,200	1	25,400	4

Table A.2: Number of Iterations until Partitioning, View Size Is 5

Reinforcement is 1000000 Fanout = 2 View Size = 5

Nb iterations untill partition	Partition Size																	
16 Use Old																		
#killed/rd	8		4		2		1		0.8		0.6		0.4		0.2		0	
Average	**2**	**2.23**	**3**	**2.13**	**8**	**1.90**	**58**	**1.77**	**180**	**1.50**	**330**	**1.73**	**1,453**	**1.60**	**19,963**	**1.30**	**2,071,224,260**	**7.00**
Std	0	2.47	1	2.10	3	1.21	50	0.97	92	1.48	249	1.11	1,707	1.30	22,010	0.88	2,978,162,733	0.71
Max	2	8	5	8	15	5	232	5	400	5	1,000	5	7,200	7	87,400	4	7,361,881,200	8
Min	1	0	2	0	3	1	6	1	100	0	100	1	100	0	1,000	0	297,688,800	6
16 Always 0																		
	8		4		2		1		0.8		0.6		0.4		0.2		0	
Average	**2**	**1.87**	**4**	**1.77**	**12**	**1.33**	**58**	**1.33**	**640**	**0.70**	**1,593**	**0.80**	**3,610**	**1.10**	**33,840**	**1.17**	**1,369,810,157**	**6.71**
Std	1	2.39	1	1.59	7	1.12	59	0.92	589	0.79	1,919	0.85	3,943	0.88	31,496	0.59	1,146,189,303	0.96
Max	3	7	7	7	32	6	288	4	2,700	3	9,300	3	18,400	5	130,900	3	3,752,802,200	8
Min	1	0	2	0	4	0	7	0	100	0	100	0	100	0	1,000	0	131,820,800	5
16 Random Copy																		
	8		4		2		1		0.8		0.6		0.4		0.2		0	
Average	**2**	**1.80**	**4**	**1.93**	**9**	**1.53**	**71**	**1.53**	**1,033**	**1.43**	**2,397**	**1.40**	**8,470**	**1.30**	**31,030**	**1.13**	**1,192,736,314**	**6.71**
Std	0	2.19	1	1.76	5	1.17	87	1.50	876	1.17	1,893	1.07	10,433	0.79	33,576	0.51	803,591,453	0.95
Max	3	6	7	7	26	4	382	6	3,300	5	7,600	4	48,700	3	142,100	2	2,558,487,700	8
Min	1	0	2	0	3	0	3	0	100	0	100	0	500	0	4,000	0	266,455,100	6
64 Use Old																		
#killed/rd	8		4		2		1		0.8		0.6		0.4		0.2		0	
Average	**5**	**1.93**	**30**	**1.30**	**952**	**1.23**	**19,718**	**1.07**	**47,940**	**1.40**	**180,497**	**1.23**	**414,133**	**1.47**	**3,531,047**	**1.50**		
Std	1	1.39	22	0.47	1,211	0.50	19,601	0.45	47,237	0.56	191,698	0.50	345,352	0.63	3,210,243	0.78		
Max	8	5	100	2	5,770	3	94,612	3	208,800	3	801,500	3	1,565,400	3	9,983,000	4		
Min	2	1	5	1	50	1	321	0	500	1	200	1	27,900	1	34,300	1		
64 Always 0																		
	8		4		2		1		0.8		0.6		0.4		0.2		0	
Average	**6**	**1.73**	**32**	**1.23**	**767**	**1.13**	**20,687**	**1.17**	**182,730**	**1.07**	**351,687**	**1.07**	**977,020**	**1.30**	**2,964,490**	**1.43**		
Std	2	1.64	30	0.90	868	0.43	18,329	0.38	124,581	0.25	330,291	0.45	1,087,982	0.70	2,918,826	0.57		
Max	10	9	109	5	4,311	2	66,979	2	575,900	2	1,372,400	2	4,132,600	4	12,237,600	2		
Min	3	1	4	0	33	0	562	1	38,200	1	17,400	0	15,300	1	20,700	0		
64 Random Copy																		
	8		4		2		1		0.8		0.6		0.4		0.2		0	
Average	**5**	**1.53**	**36**	**1.03**	**954**	**1.13**	**24,960**	**1.20**	**195,450**	**1.23**	**354,397**	**1.13**	**1,115,820**	**1.17**	**5,474,583**	**1.43**		
Std	2	0.97	38	0.18	879	0.43	23,082	0.41	180,922	0.50	385,586	0.43	1,115,332	0.38	5,680,140	0.73		
Max	10	5	143	2	3,683	3	102,325	2	755,800	3	2,057,500	3	4,300,700	2	24,987,100	4		
Min	3	1	4	1	15	1	615	1	1,300	1	24,800	1	19,100	1	162,900	1		
256 Use Old																		
#killed/rd	8		4		2		1		0.8		0.6		0.4		0.2		0	
Average	**337**	**1.03**	**9,210**	**1.10**	**166,969**	**1.20**	**1,747,743**	**1.50**	**3,123,503**	**1.37**	**8,088,853**	**1.71**						
Std	258	0.18	8,069	0.31	178,607	0.41	1,670,901	0.57	4,151,210	0.61	6,984,553	0.59						
Max	1,240	2	40,068	2	706,202	2	6,622,275	3	19,307,600	3	23,854,400	3						
Min	4	1	186	1	1,690	1	1,374	1	22,500	1	575,100	1						
256 Always 0																		
	8		4		2		1		0.8		0.6		0.4		0.2		0	
Average	**291**	**1.13**	**10,394**	**1.10**	**214,944**	**1.30**	**1,875,577**	**1.43**	**3,738,552**	**1.40**	**4,945,121**	**1.30**	**18,179,312**	**1.53**	**68,578,465**	**1.67**		
Std	356	0.35	7,648	0.48	200,111	0.53	2,042,079	0.57	3,177,265	0.62	5,782,676	0.53	15,764,161	0.82	60,950,961	0.58		
Max	1,795	2	28,758	3	746,044	3	8,716,529	3	11,523,365	3	26,863,083	3	52,199,514	4	134,013,616	2		
Min	7	1	169	0	8,625	1	119,948	1	116,219	0	42,714	1	281,813	1	13,417,618	1		
256 Random Copy																		
	8		4		2		1		0.8		0.6		0.4		0.2		0	
Average	**343**	**1.10**	**8,984**	**1.17**	**158,598**	**1.17**	**2,033,187**	**1.23**	**5,728,317**	**1.30**	**18,827,625**	**1.13**						
Std	272	0.40	10,622	0.38	136,985	0.38	2,809,876	0.50	4,694,131	0.53	13,015,366	0.35						
Max	1,313	3	54,770	2	542,895	2	13,843,823	3	18,888,300	3	42,016,300	2						
Min	73	1	566	1	3,548	1	122,964	1	638,100	1	3,021,600	1						

Table A.3: Number of Iterations until Partitioning, View Size Is 6 and Up

Reinforcement is 1000000 Fanout = 2 View Size = 6

Nb iterations untill partition	Partition Size															
16	Always 0															
#killed/rd	8		4		2		1		0.8		0.6		0.4		0.2	
Average	**2**	**2.20**	**4**	**1.80**	**10**	**1.37**	**46**	**1.27**	**440**	**0.80**	**1,077**	**1.47**	**9,257**	**1.17**	**46,107**	**1.13**
Std	1	2.41	2	2.07	5	1.40	40	0.45	403	1.27	949	1.61	8,835	0.91	44,241	0.63
Max	4	8	10	8	22	7	170	2	2,100	6	3,900	7	32,800	5	160,600	3
Min	1	0	2	0	4	0	6	1	100	0	100	0	700	0	3,300	0
64	Always 0															
	8		4		2		1		0.8		0.6		0.4		0.2	
Average	**5**	**2.00**	**47**	**1.20**	**2,323**	**1.20**	**113,893**	**1.17**	**4,425,400**	**0.83**	**18,193,000**	**0.97**	**48,031,033**	**1.07**	**239,144,278**	**1.06**
Std	2	2.98	41	0.61	2,315	0.61	96,596	0.59	3,524,794	0.46	22,673,278	0.49	69,142,239	0.25	176,797,285	0.73
Max	13	17	182	4	9,125	4	393,112	3	14,391,000	2	86,989,000	2	297,101,000	2	575,418,000	3
Min	3	0	8	1	111	1	3,294	0	385,000	0	1,000	0	2,781,000	1	28,534,000	0
256	Always 0															
	8		4		2		1		0.8		0.6		0.4		0.2	
Average	**821**	**1.03**	**56,545**	**1.07**	**2,225,416**	**1.17**	**34,561,363**	**1.29**								
Std	738	0.18	65,709	0.25	2,765,618	0.38	19,974,928	0.49								
Max	3,169	2	340,433	2	11,532,002	2	77,749,366	2								
Min	20	1	1,654	1	42,538	1	18,257,079	1								

Reinforcement is 1000000 Fanout = 2 View Size = 7

16	Always 0															
#killed/rd	8		4		2		1		0.8		0.6		0.4		0.2	
Average	**2**	**1.87**	**5**	**1.60**	**10**	**2.07**	**27**	**1.27**	**413**	**1.17**	**920**	**0.87**	**2,737**	**1.00**	**24,570**	**1.03**
Std	0	2.52	1	1.73	5	1.60	15	0.78	315	1.29	1,126	0.97	1,805	0.69	22,550	0.85
Max	3	8	8	6	21	7	66	4	1,300	6	5,600	5	6,200	3	87,100	5
Min	2	0	2	0	4	1	9	0	100	0	100	0	400	0	200	0
64	Always 0															
	8		4		2		1		0.8		0.6		0.4		0.2	
Average	**6**	**1.73**	**50**	**1.13**	**4,760**	**0.97**	**439,409**	**1.20**	**21,282,333**	**0.60**	**82,927,200**	**1.00**	**189,842,500**	**1.00**		
Std	2	1.23	38	0.51	4,521	0.41	383,359	0.41	20,896,985	0.50	89,466,181	0.45	152,197,993	0.67		
Max	11	6	149	3	15,949	2	1,525,117	2	67,863,000	1	331,606,000	2	458,226,000	2		
Min	3	1	6	0	362	0	7,048	1	763,000	0	704,000	0	6,129,000	0		
256	Always 0															
	8		4		2		1		0.8		0.6		0.4		0.2	
Average	**2,804**	**1.00**	**245,116**	**1.10**	**24,177,934**	**1.14**										
Std	2,561	0.26	261,875	0.31	21,845,081	0.35										
Max	8,732	2	1,232,894	2	87,377,684	2										
Min	53	0	5,624	1	443,858	1										

View Size = 8 View Size = 10 View Size = 11 View Size = 12

256	Always 0								1,024				2,048		4,096	
#killed/rd	16		8		4		2		32		16		64		128	
Average	**37**	**1.27**	**5,153**	**1.10**	**715,496**	**1.20**	**128,737,713**	**1.00**	**40,520**	**1.03**	**7,163,336**	**1.00**	**50,651**	**1.02**	**71,764**	**1.01**
Std	31	1.11	6,254	0.31	684,858	0.48	119,711,978	0.00	42,382	0.18	4,121,391	0.00	47,465	0.14	75,140	0.12
Max	117	7	28,072	2	2,381,973	3	269,246,107	1	179,200	2	13,905,400	1	229,000	2	513,800	2
Min	7	1	155	1	17,919	1	9,350,675	1	3,400	1	1,298,400	1	700	1	600	1

BIBLIOGRAPHY

[1] D. Agrawal, A. El Abbadi, and R. Steinke. Epidemic algorithms in replicated databases. In *Proc. 16th ACM Symp. on Principles of Database Systems*, pages 161–172, 1997.

[2] J. Aspnes, Z. Diamadi, and G. Shah. Fault-tolerant routing in peer-to-peer systems. In *Proc. 21st ACM Symp. on Principles of Distributed Computing*, pages 223–232, 2002.

[3] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Message multicasting in heterogeneous networks. *SIAM J. on Computing*, 30:347–358, 2001.

[4] K. Birman, A. Demers, I. Gupta, D. Kostoulas, and D. Psaltoulis. Practical algorithms for size estimation in large and dynamicgroups. In *Proc. 21st ACM Symp. on Principles of Distributed Computing*, 2002.

[5] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17:41–88, 1999.

[6] BitTorrent. http://www.bittorrent.com.

[7] D. Boyd and J. M. Steele. Random exchanges of information. *Journal of Applied Probabilities*, 16:657–661, 1979.

[8] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Exploiting network proximity in peer-to-peer overlay networks. In Ozalp Babaoglu, Ken Birman, and Keith Marzullo, editors, *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, pages 52–55, June 2002.

[9] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), October 2002.

[10] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scalable application-level anycast for highly dynamic groups. In *Networked Group Communication, Fifth International COST264 Workshop (NGC'2003)*, September 2003.

[11] Miguel Castro, Michael B. Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, and Alec Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Proc. 22nd IEEE INFOCOM Conference*, April 2003.

[12] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology. In *ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, pages 20–41, 2001.

[13] F. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[14] Fan Chung. Laplacian and the cheeger inequality for directed graphs. *Annals of Combinatorics*, 9:1–19, 2005.

[15] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 311–320, 2000.

[16] C. Cooper, M. Dyer, and C. Greenhill. Sampling regular graphs and a peer-to-peer network. In *Proc. 16th ACM Symp. on Discrete Algorithms*, 2005.

[17] X. Defago, A. Schiper, and P. Urban. Totally ordered broadcast and multicast algorithms: a comprehensive survey. Technical Report DSC/2000/036, Dept. of Communication Systems, École Polytechnique Fédérale de Lausane, Switzerland, 2000.

[18] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. 7th ACM Symp. on Operating Systems Principles*, pages 1–12, 1987.

[19] D. P. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *Random Structures and Algorithms*, 13(2):99–124, 1998.

[20] eDonkey2000 Overnet. http://www.edonkey2000.com.

[21] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proc. 5th Intl. Conf. on Mobile Computing and Networking*, pages 263–270, 1999.

[22] P. Eugster, R. Guerraoui, S. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4), November 2003.

[23] U. Feige, D. Peleg, P. Raghavan, and E. Upfal. Randomized broadcast in networks. *Random Structures and Algorithms*, 1:447–460, 1990.

[24] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. In *Proc. 32nd ACM Symp. on Theory of Computing*, pages 218–226. ACM Press, 2000.

[25] Freenet. http://freenet.sourceforge.net.

[26] A. Frieze and G. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10:57–77, 1985.

[27] A.J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Hi-scamp: Self-organizing hierarchical membership protocol. In *Proc. SIGOPS, European Workshop*, September 2002.

[28] A.J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Network awareness and failure resilience in self-organising overlay networks. In *Proc. 22nd IEEE Symp. on Reliable Distributed Systems*, 2003.

[29] A.J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2), February 2003.

[30] A.J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3), March 2003.

[31] Gnutella. http://www.gnutella.com.

[32] A. Gupta, B. Liskov, and R. Rodrigues. One hop lookups for peer-to-peer overlays. In *Proc. of 9th Workshop on Hot Topics in Operating Systems*, pages 7–12, 2003.

[33] A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. In *Proc. 1st Networked Systems Design and Implementation*, pages 113–126, 2004.

[34] I. Gupta, K. P. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *Proc. 2nd International Workshop on Peer-to-Peer Systems*, pages 160–169, 2003.

[35] I. Gupta, A.-M. Kermarrec, and A. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *21st Symp. on Reliable Distributed Systems*, pages 180–189, 2002.

[36] I. Gupta, R. van Renesse, and K. Birman. Scalable fault-tolerant aggregation in large process groups. In *Proc. Conf. on Dependable Systems and Networks*, pages 433–442, 2001.

[37] Z. J. Haas, J. Y. Halpern, and E L. Li. Gossip-based ad hoc routing. In *Proc. 21st IEEE INFOCOM Conference*, 2002.

[38] S. Hedetniemi, S. Hedetniemi, and A. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1988.

[39] R. van Renesse I. Gupta, K. Birman. Fighting fire with fire: using randomized gossip to combat stochastic scalability limits. *Special Issue Journal Quality and Reliability Engineering International: Secure, Reliable Computer and Network Systems*, 18(3):165–184, May/June 2002.

[40] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proc. 22nd Intl. Conf. on Distributed Computing Systems*, 2002.

[41] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Proc. 5th ACM/IFIP/USENIX International Middleware Conference*, 2004.

[42] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast computing. Technical report, Vrije Universiteit Amsterdam, November 2003.

[43] Márk Jelasity and Ozalp Babaoglu. T-Man: Fast gossip-based construction of large-scale overlay topologies. Technical Report UBLCS-2004-7, University of Bologna, Department of Computer Science, Bologna, Italy, May 2004. `http://www.cs.unibo.it/techreports/2004/2004-07.pdf`.

[44] J. Kahn, R. Katz, and K. Pister. Next century challenges: Mobile networking for 'smart dust'. In *Proc. 5th Intl. Conf. on Mobile Computing and Networking*, pages 271–278, 1999.

[45] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *Proc. 41st IEEE Symp. on Foundations of Computer Science*, pages 565–574, 2000.

[46] R. M. Karp, E. Upfal, and A. Wigderson. The complexity of parallel search. *Journal of Computer and System Science*, 36(2):225–253, 1988.

[47] D. Kempe, A. Dobra, and J. Gehrke. Computing aggregate information using gossip. In *Proc. 44th IEEE Symp. on Foundations of Computer Science*, pages 482–491, 2003.

[48] D. Kempe and J. Kleinberg. Protocols and impossibility results for gossip-based communication mechanisms. In *Proc. 43rd IEEE Symp. on Foundations of Computer Science*, pages 471–480, 2002.

[49] D. Kempe, J. Kleinberg, and A. Demers. Spatial gossip and resource location protocols. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 163–172, 2001.

[50] S. Krishnamurthy, S. El-Ansary, E. Aurell, and S. Haridi. A statistical theory of chord under churn. In *Proc. 4st International Workshop on Peer-to-Peer Systems*, 2005.

[51] J. Li, J. Stribling, T. Gil, R. Morris, and M. F. Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proc. 3rd International Workshop on Peer-to-Peer Systems*, pages 87–99, 2004.

[52] D. Liben-Nowell, H. Balakrishnan, and D. R. Karger. Analysis of the evolution of peer-to-peer systems. In *Proc. 21st ACM Symp. on Principles of Distributed Computing*, pages 233–242, 2002.

[53] M. Lin, K. Marzullo, and S. Masini. Gossip versus deterministic flooding: Low message overhead and high reliability for broadcasting on small networks. Technical Report CS99-0637, UC San Diego, 1999.

[54] M.-J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *Proc. 3rd European Dependable Computing Conference*, pages 364–379, 2000.

[55] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Survey and Tutorial*, March 2004.

[56] J. Luo, P. Eugster, and J. Hubaux. Route driven gossip: Probabilistic reliable multicast in ad hoc networking. In *Proc. 22nd IEEE INFOCOM Conference*, 2003.

[57] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. In *Proc. 5th Symp. on Operating Systems Design and Implementation*, 2002.

[58] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proc. 18th Intl. Conf. on Data Engineering*, 2002.

[59] S. R. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proc. 2002 ACM SIGMOD Intl. Conference on Management of Data*, pages 49–60, 2002.

[60] P. Mahlmann and C. Schindelhauer. Peer-to-peer networks based on random transformations of connected regular undirected graphs. In *Proc. 17th ACM Symp. on Parallel Algorithms and Architectures*, 2005.

[61] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proc. 21st ACM Symp. on Principles of Distributed Computing*, 2002.

[62] Dahlia Malkhi. Personal communication, June 2005.

[63] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. 1st International Workshop on Peer-to-Peer Systems*, May 2002.

[64] R. Melamed and I. Keidar. Araneola: A scalable reliable multicast system for dynamic environments. In *Proc. 3rd IEEE International Symp. on Network Computing and Applications*, 2004.

[65] J. W. Moon. Random exchanges of information. *Nieuw Archief voor Wiskunde*, 3:246–249, 1972.

[66] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1990.

[67] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. In *Proc. 42nd IEEE Symp. on Foundations of Computer Science*, pages 492–499, 2001.

[68] B. Pittel. On spreading a rumor. *SIAM J. Applied Math.*, 47:213–223, 1987.

[69] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM Conference*, pages 161–172, 2001.

[70] R. Ravi. Rapid rumor ramification: Approximating the minimum broadcast time. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pages 202–213, 1994.

[71] R. Rodrigues and C. Blake. When mutli-hop peer-to-peer routing matters. In *Proc. 3rd International Workshop on Peer-to-Peer Systems*, 2004.

[72] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. 18th IFIP/ACM Intl. Conf. on Distributed Systems Platforms (Middleware 2001)*, pages 329–350, 2001.

[73] J. Saia, A. Fiat, S. Gribble, A. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In *Proc. 1st Intl. Workshop on Peer-to-Peer Systems*, 2002.

[74] K. Shen. Structure management for scalable overlay service construction. In *Proc. 1st USENIX/ACM Symp. on Networked Systems Design and Implementation*, 2004.

[75] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM Conference*, pages 149–160, 2001.

[76] R. van Renesse. Scalable and secure resource location. In *Proc. 33rd Hawaii Intl. Conf. on System Sciences*, 2000.

[77] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2), May 2003.

[78] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure-detection service. In *Proc. IFIP Intl. Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 55–70, 1998.

[79] E. Vollset and P. Ezhilchelvan. A survey of reliable broadcast protocols for mobile ad-hoc networks",. Technical Report CS-TR-792, University of Newcastle, 2003.

[80] S. Voulgaris, D. Gavidia, and M. van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, to appear, 2005.

[81] Jun Xu. On the fundamental tradeoffs between routign table size and network diameter in peer-to-peer networks. In *Proc. 22nd IEEE INFOCOM Conference*, 2003.

[82] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.